1) Indicate whether f=O(g) or f=Ω(g) or both(in which case f=θ(g)) along with your justification.

   f (n) = n1/2 g (n) = 8^(log n2)

   F(n) < g(n), hence f(n)=O(g(n)

2) Arrange the following functions in the order of complexity: (You can simply state the function names in the order you decide if you would like)

   Log n < n log n < n \sqrt n < n^10 < 2^n

   OR

   f3 < f5 < f1 < f2 < f4

3) Write the complexity of the Merge Sort algorithm using the Master theorem. You need to explain the breakdown of the Merge Sort algorithm in order to come up with the final complexity.

   In merge sort, we divide the problem into 2 subproblems and solve each subproblem to obtain the final sorted
   No of subproblems: 2
   Size of each subproblem: n/2
   Time for merge: constant time O(1)

   By taking the above into consideration a and b turns out to be 2 and 2 respectively and d = 0.
   T(n) =2T(n/2)+O(n^1)
   d=log a = log 2 = 1

   Hence T(n) = 2T(n/2) + O(n) which results in O(n log n)

4) A) MST application: Consider a telecommunications company wants to lay cable in this neighborhood. Nodes are the houses, the edges are the paths between them, and the edge weights are the maintenance cost. The goal is to find the cheapest way to lay cable while connecting all houses. MSTs can help with that.

   B) Kruskal's: Repeatedly add the next *best edge* that does not produce a cycle. Ties are broken arbitrarily. The most dominant cost in Kruskal's algorithm is to sort the edges, which takes $O(|E| \log |E|)$ time. It is preferred to have a tight upper bound with V instead of E. In a complete graph (i.e. where each node is connected to every other node), we have:

$$|E| \leq |V|^2$$
$$\log|E| \leq \log|V|^2$$
$$\log|E| \leq 2 \log |V|$$

Hence, total running time complexity is:

$$O(|E| \log |V|)$$

Prim's algorithm: Choose any vertex randomly. Select the best edge from the set of edges connected to one of the seen vertices. Cannot produce a cycle. Same complexity as Kruskal's if Binary min-heaps are used.

5) Given two strings S and T, find the number of times T occurs in the first string S (it can be continuous or discontinuous as a subsequence)

Example 1:
Input:
S: sequence
T: sue
Output: 2 because: sequence, sequence

Example 2:
Input:
S: distinct
T: it
Output: 3 because: distinct, distinct, distinct

What is the naive algorithm to do this? What is the complexity of the overall naive algorithm?

2^n for a string of length n

6) Write a dynamic programming solution for the Longest Palindromic Subsequence problem (https://www.geeksforgeeks.org/longest-palindromic-subsequence-dp-12/)

```
for (i = 0; i < n; i++)
     L[i][i] = 1;


for (cl=2; cl<=n; cl++)
    {
        for (i=0; i<n-cl+1; i++)
        {
            j = i+cl-1;
            if (str[i] == str[j] && cl == 2)
               L[i][j] = 2;
            else if (str[i] == str[j])
               L[i][j] = L[i+1][j-1] + 2;
            else
               L[i][j] = max(L[i][j-1], L[i+1][j]);
        }
    }
```