OLD Homework problem (not for submission):
MST.

Let \( G \) be an undirected graph with \( n \) vertices and \( m \) edges with weights on the edges (here \( m \geq n \)). Furthermore, assume that the given weights \( w(\cdot) \) on the edges are all distinct.

28.A. (30 pts.) For a path \( \pi \) in the graph, its bottleneck price \( \beta(\pi) \) is the maximum weight of an edge on \( \pi \). Formally, \( \beta(\pi) = \max_{e \in \pi} w(e) \). The bottleneck distance for two vertices \( u, v \in V(G) \) is \( \beta(u, v) = \min_{\pi \in \Pi(u,v)} \beta(\pi) \), where \( \Pi(u, v) \) is the set of all paths between \( u \) and \( v \) in \( G \).

Describe how to build a data-structure, using \( O(n) \) space, such that given a query pair of vertices \( u, v \in V(G) \), one can compute \( \beta(u, v) \) in \( O(n) \) time. Describe the query algorithm that computes the distance, and prove its correctness (i.e., that the result it returns is correct). What is the construction time of the data-structure?

[Harder but doable by you. Not for submission (and we will not provide a solution): Show how to build a data-structure that uses \( O(n \log n) \) space, and answers such queries in \( O(\log n) \) time.]

28.B. (10 pts.) Consider computing the MST \( T_1 \) from \( G \), removing the edges of \( T_1 \) from \( G \), computing an MST \( T_2 \) of the remaining graph, and continuing in this fashion till the graph is disconnected, and let \( T_1, \ldots, T_k \) be the extracted spanning trees.

Computing such trees can be useful for robustness – if the first MST fails, you have a tree which is almost as good as backup (i.e., \( T_2 \)), and so on.

Clearly, this sequence can be computed in \( O(k(n \log n + m)) \) time (how?). We are interested in a faster algorithm (for example, think about the case that \( k = \Omega(\sqrt{n}) \)).

To this end, let \( e_1, \ldots, e_m \) be the edges of \( G \) in increasing sorted order by weight. Let \( L_1(t) = \{e_1, \ldots, e_t\} \) be the set of the first \( t \) edges in this order. Let \( F_1(t) \) be the set of edges used by the spanning forest computed by the Kruskal algorithm after inserting the edges of \( L_1(t) \).

More generally, for \( i > 0 \), let \( L_{i+1}(t) = L_i(t) \setminus F_i(t) \), and let \( F_{i+1}(t) \) be the edges of the spanning forest of \( L_{i+1}(t) \) as computed by the Kruskal algorithm when executed on the set of edges \( L_{i+1}(t) \).

Prove that \( F_1(m), \ldots, F_k(m) \) are the edges of the trees \( T_1, \ldots, T_k \), respectively.

28.C. (10 pts.) In the context of (B). Prove, that if two vertices \( u, v \) are in the same connected component of the graph \( (V, F_i(t)) \), for some time \( i \), then they are in the same connected component of \( (V, F_j(t)) \), for all \( j < i \).
28.D. (20 pts.) Let $D_i(t)$ be the union-find data-structure used by the Kruskal algorithm, defined over the $V(F_i(t))$, after handling the edges of $L_i(t)$. Assume that you have $D_1(t), D_2(t), \ldots$, and the edge $e_{t+1} = u'_{t+1}v'_{t+1}$. Show how to compute, in $O(\alpha(n) \log n)$ time, the minimal index $j > 0$, such that the two vertices $u'_{t+1}$ and $v'_{t+1}$ of $e_{t+1}$ appear in two different connected components of $F_j(t)$.

Here, assume $\alpha(n)$ is a bound on the time it takes to perform a single operation in the union-find data-structure.

28.E. (30 pts.) Using the above, show how to compute the spanning trees $T_1, \ldots, T_k$, from part (B), in $O(m \alpha(n) \log n)$ time. Be careful about the details – the value of $k$ is not given to you, and you might need to create new sets (of a single element) in the union-find data-structures on the fly.

Do not use hashing or dictionary data-structures in solving this problem.

29. (100 pts.) OLD Homework problem (not for submission):
Undecidable.

For each of the following languages, either prove that it is undecidable (by providing a detailed reduction from a known undecidable language), or describe an algorithm that decides this language – your description of the algorithm should be detailed and self contained. (Note, that you cannot use Rice Theorem in solving this problem.)

29.A. (25 pts.) $L_1 = \{\langle D, N \rangle \mid L(D) = L(N), \text{where } D \text{ is a DFA, and } N \text{ is an NFA}\}$

29.B. (25 pts.) $L_2 = \{\langle M, D \rangle \mid L(M) = L(D), \text{where } M \text{ is a Turing machine, and } D \text{ is a DFA}\}$

29.C. (25 pts.) $L_3 = \{\langle D \rangle \mid L(D) \text{ is finite, where } D \text{ is a DFA}\}$.

29.D. (25 pts.) $L_4 = \{\langle M \rangle \mid L(M) \text{ is finite, where } M \text{ is a Turing machine}\}$. 
