Prob. 1	Prob. 2	Prob. 3	Prob. 4

Problem 1.

1. To decide whether there exists k edge-disjoint paths, we can define the capacity of each edge as 1. Thus the flow will only go through an edge at most 1 time. Then we push the maximum flow possible from s. It means a flow of $n = degree(s)$. So the maximum number of edge-disjoint paths is bounded by n. Then we see how many of the incoming edges of t will be used and check if it's bigger or equal to k. It means we pass through at most $|E|$ edges. So this algorithm runs in polynomial time.

2. To check if there are k node-disjoint paths, we can use the same algorithm as before but every time we reach a node (which is not t) by a given edge, we need to set the capacity of its other incoming edges (obviously if the in-degree of the node is bigger than 1) to 0. So a node can be accessed only 1 time. Then we see how many of the incoming edges of t will be used and check if it's bigger or equal to k. The algorithm runs in polynomial time since we visit at most $|V|$ nodes but we need also to change capacity of at most $|V|$ incoming edges.

3. Let's define E_1 as the set of all edges from s to u and E_2 the set of edges from u to t. Since it's a directed graph, E_1 and E_2 are disjoint. It means one of the k paths from s to u will not share any edge with one of the k paths from u to t. So from s to t we have at least k paths ("at least" because there may exist path from s to t that doesn't pass by u).

4. The counter example in Figure 1 shows that a directed graph may have k paths from s to u and k paths from u to t but there are less paths from s to t.

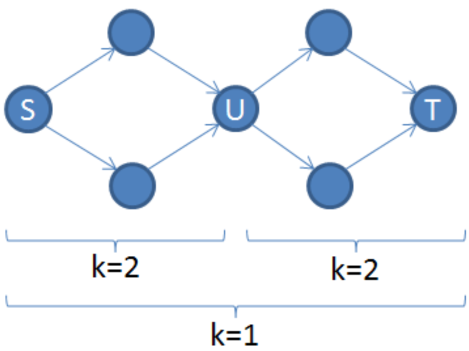

Figure 1: A counter example

Problem 2.

Problem 3.

Our approach to the problem is the following:
We create a source node, and to this we connect n nodes each representing one of the n persons sharing the wash machine (lets call those nodes P_i , i ranging from 1 to n). The connecting edges between the source and the P_i nodes initially have all a capacity of l. Then we create a sink node and to this node we connect m nodes that represent the available time slots (lets call those nodes T_j , j ranging from 1 to m). All edges connecting the time slots with the sink have a capacity of 1. Then we connect each P_i node with all the T_j nodes that are included to this person's candidate list, again with edges of unitary capacity. When this process is complete, we have our network.

We apply to the network that we created with the above mentioned procedure, the Ford-Fulkerson algorithm to find the maximum flow of the network. If we can get a maximum flow of $n*l$ that implies that there is a minimum valid assignment (meaning that every person gets the mimimum time slots among the ones in his list). If there is such an assignment and if k is greater than $l*n$ we proceed as following:
We update each node's capacity to h, and then we apply an algorithm to find $k-n*l$ augmenting paths within our network, so that we get a network with flow equal to k. However, we apply the algorithm using one condition: that there can be no paths that go from a P_i node to the source and then back to a P_i node because there is a chance that then the corresponding edge's flow may be decreased below l, a fact that is undesirable according to the problem description.
If we find the desired number of augmenting paths then our goal is fulfilled otherwise, it cannot be reached. The cost of the algorithm is the sum of the cost of the maximum flow algorithm and the augmenting paths algorithm. Using the Ford-Fulkerson algo for the maximum flow, and the Edmonds-Karp for the augmenting paths, we get a complexity of O($V*$max(flow))+O(VE^2) = O($V*n*l$)+O($V*E^2$) which is polynomial since the number of edges is bound by some multiple of n.

Problem 4.

This problem is a simplification of *Convex Hull* problem because here we just need to traverse the points, no need to check if traversing is going clockwise or not. Algorithm is the following for n points:

- Find the initial point p_i which has *minimum* y, finding minimum of n points is bound by $O(n)$
- Sort the remaining points in ascending order according to polar angle that they made with p where it is bound by $O(n \log n)$ because we only spend $O(1)$ time for each polar angle calculation, in total, that results in $O(n)$ which is dominated by sorting n points.
- Traverse the points in the order and put an edge between the current point p_c and the following point p_f. Since we are traversing all points, this is $O(n)$.
- At the end, put one more edge between p_c and p_i to close the loop and create a polygon. This operation is just a constant time operation, $O(1)$.

In short, we have:

$$\begin{aligned} \textit{Total Cost} &= \textit{Find}\, p_i + \textit{Calculate polar angles with } p_i + \textit{Sort points} + \textit{Close loop} \\ &= O(n) + O(n) + O(n \log n) + O(1) \leq O(n \log n) \end{aligned}$$