Problem 1 (Graded by Rajendra):

Algorithm:
Since the given numbers are sorted in increasing order, let us start from the smallest number and try to fit in as many real numbers in this set, such that the difference between the smallest and the largest number in this range is <= 1.

If not all real numbers could be accommodated because of the condition stated above, then start a new set with the next real number in line and fill it in the same manner.

Go on doing this until the whole set of real numbers provided is exhausted.

Proof of optimality:
Let us represent the set “i” of unit length of our greedy approach by [s_i, b_i]

So let the solution given by our greedy approach (S_G) and by some other optimal approach(S_O) as follows :

\[
\begin{array}{cccccccc}
1 & 2 & \ldots & k & \ldots & m \\
S_G & \rightarrow & [s_1, b_1] & [s_2, b_2] & \ldots & [s_k, b_k] & \ldots & [s_m, b_m] \\
S_O & \rightarrow & [S_1, B_1] & [S_2, B_2] & \ldots & [S_k, B_k] & \ldots & [S_m, B_m] \\
\end{array}
\]

Let us assume the sets in both solutions are similar till j < k. They differ in the set k.

Thus s_k = S_k since the given set of numbers is sorted.
So the k^{th} set differs only in the values of b_k and B_k.

There are two cases now:

1) b_k < B_k \Rightarrow This case is not possible since our approach is greedy and we try to fill in as many numbers into this set k till we are fine with the condition (b_k - s_k) <= 1 . So if such B_k existed then we would have surely included it in the set k of our greedy solution.

2) b_k > B_k \Rightarrow In this case move numbers from the sets j > k in S_O till B_k = b_k . This will not result in an absurd solution as we already know a solution is possible when the k^{th} set of the solution S_G has the biggest number equal to b_k .

So repeating the procedure as we did in condition (2), we can show that the optimal solution S_O becomes equal to S_G. Hence our greedy approach will give an optimal solution.

Complexity analysis: O(n)
Problem 2 (Graded by Manna):

No, the Greedy method will not work. Here is a counterexample:

Let’s try the Greedy method on the problem depicted on a time scale. The line fragments represent the working schedules for various people (represented by number).

![Time Scale Diagram]

Here, if you choose the employee with the least clashes (Employee 1), it will eliminate 2 & 3, and result in a picture with two clear columns.

Now, all the employees have the same number of clashes. So, you can only choose one from each column (say 4 & 5). This gives maximum size of 3 = {1, 4, 5}.

However, it’s clearly seen that the optimal solution $S = \{6, 2, 3, 7\}$ is of size 4. So, Greedy does not work here.

Problem 3 (Graded by Manas):

Since the set of vertices are the same and in the resultant graph $G'$ and there is exactly one path between any two nodes, it’s a spanning tree.

However, we also stipulate that the cost of the edges that are not there in the resulting graph $G'$ are to be minimized by your algorithm. In other words, the cost of the edges that are there in the resulting graph is to be maximized. Which means, your algorithm should give a Maximum Spanning Tree of $G$.

There are many ways to do this, one of them using the Kruskal’s method. Just change the sign of the edge weights in the original graph, so that the lightest becomes the heaviest and vice versa. Then run the Kruskal’s algorithm to find the Minimum spanning tree of the modified graph ($G''$), which would be the same as the Maximum Spanning Tree of the original graph ($G'$). The complexity is same as that of Kruskal’s algorithm.