1. **Scheduling jobs intervals with penalties**:

For each 1 ≤ 𝑖 ≤ 𝑛 job 𝑗𝑖 is given by two numbers 𝑑𝑖 and , where 𝑑𝑖 is the deadline and 𝑝𝑖 is the penalty. The length of each job is equal to 1 minute. We want to schedule all jobs, but only one job can run at any given time. If job i does not complete on or before its deadline , we should pay its penalty 𝑝𝑖 . Design a greedy algorithm to find a schedule which minimizes the sum of penalties.

**Solution:**

**Observation:** We can assume that all jobs finish after 𝑛 minutes. Suppose not. So there is an empty minute starting at say 0 ≤ 𝑘 ≤ 𝑛 − 1 (where no job is scheduled) and there is a job 𝑗𝑖 for some 1 ≤ 𝑖 ≤ 𝑛 which is scheduled some time after n minutes. If we schedule job 𝑗𝑖 to start at minute 𝑘, then this can only be a better solution since everything remains same except 𝑗𝑖 might now be able to meet its deadline.

We assign time intervals 𝑀𝑖 for 1 ≤ 𝑖 ≤ 𝑛 where 𝑀𝑖 starts at minute 𝑖 − 1 and ends at minute 𝑖. The greedy algorithm is as follows:

* Arrange the jobs in decreasing order of the penalties 𝑝1 ≥ 𝑝2 ≥ ⋯ ≥ 𝑝𝑛 and add them in this order
* To add job 𝑗𝑖 ,
* if any time interval 𝑀𝑙 is available for 1 ≤ 𝑙 ≤ 𝑑𝑖 , then schedule 𝑗𝑖 in the last such available interval.
* else schedule 𝑗𝑖 in the first available interval starting backwards from 𝑀𝑛

2. Given a set {x1 ≤ x2 ≤ … ≤ xn } of points on the real line, determine the smallest set of unit-length closed intervals (e.g. the interval [1.25,2.25] includes all xi such that {1.25 ≤ xi ≤ 2.25} ) that contains all of the points. Give the most efficient algorithm you can to solve this problem, prove it is correct and analyze the time complexity.

**Solution:**

The greedy algorithm we use is to place the first interval at [x1, x1+ 1], remove all points in [x1, x1+ 1] and then repeat this process on the remaining points. Clearly this is an O(n) algorithm.

We can prove it is correct by showing the greedy choice and optimal substructure properties.

**Greedy Choice Property:** Let S be an optimal solution. Suppose S places its leftmost interval at [x, x + 1]. By definition of our greedy choice x <= x1 since it puts the first point as far right as possible while still covering x1. Let S’ be the scheduled obtained by starting with S and replacing [x, x + 1] by[x 1, x1+ 1]. We now argue that all points contained in [x, x + 1] are covered by[x1, x1+ 1]. The region covered by[x, x + 1] which is not covered by[x1, x1+ 1] is [x, x1) which is the points from x up until x1 (but not including x1). However, since x1 is the leftmost point there are no points in this region. (There could be additional points covered by[x+1, x1+ 1] that are not covered in [x, x + 1] but that does not affect the validity of S’ ). Hence S’ is a valid solution with the same number of points as S and hence S’ is an optimal solution.

**Optimal Substructure Property:** Let P be the original problem with an optimal solution S. After including the interval [x1, x1+ 1], the subproblem P’ is to find an solution for covering the points to the right of x 1+ 1. Let S’ be an optimal solution to P’. Since, cost(S) = cost(S’) + 1, clearly an optimal solution to P includes within it an optimal solution to P’.

3. You are the manager in a ﬁrm where the length of working time and the start time of work is diﬀerent for diﬀerent employees. For example, person X works every day from 8 to 11 AM, person Y from 9 AM to 1 PM, person Z from 2 to 10 PM etc. Your task is to create a workforce consisting of maximum number of people possible with non-overlapping work hours (if a person leaves at 9 AM and another starts at 9 AM, that is not an overlap). You use the following Greedy strategy:

1. Choose the person (say X) with least number of working hour over- laps with other workers.
2. Eliminate all people having working hour overlaps with X.
3. Choose the next person with the least number of overlaps with the remaining people, and so on.

Does this strategy yield the set of maximum people? If yes, prove it. Otherwise, just give a counterexample.

**Solution:**

*No, the given greedy algorithm will not work. Many counter examples can be found.*

4. (CLRS p.436) 16.3-2 - Prove that a binary tree that is not full cannot correspond to an optimal prefix code.

**Solution:**

*Since it is a prefix code all the codewords have to correspond to the leaves, otherwise, some codeword would be a prefix of another codeword. Suppose the binary tree is not full, then there is some node which is not a leaf having only one child. We could do away with that single branch, and push up the child (and all it children) by one level. This would still remain a valid prefix code, each character having distinct binary codewords. Clearly this would be of less average length than the tree that we started with. So the initial tree could not have been that of an optimal prefix code.*

5. (CLRS p.436) 16.3-7 – Generalize Huffman’s Algorithm to ternary codewords (i.e. codewords using the symbols 0, 1,2).

**Solution:**

*The algorithm is very similar to the Huffman code that we have seen in the class. Pick the smallest three frequencies, join them together and create a node with the frequency equal to the sum of the three. Repeat. However, notice that every contraction reduces the number of leaves by 2 – we remove 3 nodes and add 1 back. So to make sure that we end up with just one node, we have to have an odd number of nodes to start with. If not, add a dummy node with 0 frequency to start with.*