16.1-2

The proposed solution of selecting the last activity to start (and remains compatible with previous selected tasks) belongs to Greedy approach because it tries to make the choice which looks the best at every step.

We can regard this proposal as a "reduction" from the case when we select the activity with the earliest finishing time. Like below:

If activity a_i is characterized by [$start_i$, $finish_i$] in the set S where we select the 1^{st} activity to finish, then we can regard the modified set S1 containing activities a_j that are characterized by [$finish_i$, $start_i$] (original pairs are swapped).

The optimal solution from S gets mapped to the optimal solution from S1. In order to see this, we can simulate the original algorithm on S1.

On the other hand, we can prove that an algorithm like the one below produces an optimal solution for the modified Greedy strategy.

Algorithm (start[], finish[], i)

n=i-1

while (n>0) and ($start_i$ <$finish_n$) do

    n=n-1

    If (n>0) return Algorithm(start[], finish[],n) U {a_n}

Namely, it can be proved that 1) an activity from a non-empty problem having the latest starting time is always used by some maximum sized subset of compatible activities of the aforementioned non-empty problem and 2) picking that activity leaves empty the current subproblem.

16.1-3

- Counterexample for the strategy which always choose first the activity having the least duration (compatible with previous selected jobs):

  a_1=[1,7] , a_2=[7,11], a_3=[5,8]

The suggested strategy will choose a_3 because it has the smallest duration. However, it can be easily seen that optimal strategy would be to pick a_1 and a_2 instead.

- Counterexample for the strategy which always selects the activity which overlaps with the fewest remaining activities

  a_1=[4,6], a_2= [1,3], a_3=[7,9], a_4=[3,5], a_5=[5,7], a_6=[2,4], a_7=[2,4], a_8=[6,8],a_9=[6,8]

  The suggested strategy will select a_1, a_2 and a_3, while the optimal strategy would pick a_2, a_3, a_4, and a_5.

- Counterexample for the strategy which always selects the activity having the earliest start time (compatible with previous selected tasks)

  a_1 =[1,7], a_2=[2,5], a_3=[5,7]

  The suggested strategy will only select a_1, while the optimal strategy would be picking activities a_2 and a_3.

16.2-4

This problem can be solved in at least two ways, both based on the Greedy strategy of always determining the farthest stop within m miles. He will repeat process from start to the destination.
One way to prove this leads to optimal solution is based on the "Greedy stays ahead" property. Practically, when he first selects the farthest stopping point from Grand Forks, no other strategy could have selected a point for refilling water residing further because he would have remained without water before arriving there. So, any other strategy could have selected the same first stop in the best case. Thus, our Greedy strategy stays ahead those. The same argument can be repeated for any other stop from the path, which means the Greedy strategy we proposed makes the minimum number of stops, therefore it is optimal.
Observation – the correctness of this strategy can be also proved using the exchange argument.
The aforementioned algorithm runs in O(no.of stopping points).

16.2-5

Algorithm idea & pseudo-code:

- first, we sort the points and relabel them such that we consider them reordered like $x_1 \leq \ldots \leq x_n$ (assuming the original input set is not sorted)
- S= {} //S= the set of intervals that we must find

- Let $I_j = [x_j, x_j+1]$ for every j in the [1,n] range
- While set X is not empty do

    For every x_k from the set X do

        $S = S \cup \{I_k\}$

        $X = X - \{x_l | x_l \in I_k\}$

    Endfor

Endwhile

Return S

Proof of correctness: We assume that Sopt is some optimal set of intervals such that x_1 is covered by [y,y+1], with $y<x_1$.

Because x_1 is clearly the leftmost point => there are no other points from X contained by $[y,x_1)$

So, it is possible to drop the interval [y,y+1] from Sopt and to replace it by $[x_1,x_1+1]$ in a way that the new set of intervals remains optimal (the cardinality remains the same). This means there is an optimal set of intervals containing I_1.

On the other hand, an optimal set of intervals can be obtained if we solve the subproblem with all points from $[x_1,x_1+1]$ removed. If we consider the union between this new set and $[x_1,x_1+1]$, the same Greedy approach from above still holds if we use it for the points residing at the right of x_1+1. So, the Greedy choice property can be used for a correct algorithm to the problem.

Complexity study: if the original input set is not sorted, then the outlined Greedy algorithm runs in O(nlogn) time assuming that we sort the points using a comparison-based sorting algorithm like Mergesort or Heapsort.

16.2-7

Algorithm idea: we sort the two input sets and obtain them ordered like

$a_{1 \le \ldots\ldots \le} a_n$ and $b_{1 \le \ldots\ldots \le} b_n$ (the reversed monotony only leads to an equivalent case).

The key point of this Greedy strategy is that we pair every (a_i,b_i) to maximize the payoff.

This fact can be also proved mathematically (e.g. inequalities).

However, for the current scope, we assume by contradiction this pairing does not lead to maximum payoff (exchange argument).

Therefore, there should be a pairing between a_q and b_j AND b_q and a_j. where $a_q < a_j$ and $b_j > b_q$ (thus q<j) which leads to a bigger payoff (i.e. optimal) than pairing them as stated above (a_q,b_q) and (a_j,b_j) (the rest of pairs are the same in both solutions). Let's say the two payoffs are P1 and P2 respectively.

Then $P1/P2 = (a_q^{bj}a_j^{bq})/(a_q^{bq}a_j^{bj}) = (a_q/a_j)^{bj-bq} < 1$ (the rest of the components simplify since they are the same) => P1 < P2 =>contradiction is reached because P1 cannot be optimal as assumed.

Therefore, the proposed strategy of pairing (a_i,b_i) is optimal.

Complexity study: sorting takes O(nlogn), thus the algorithm runs in O(nlogn)+O(nlogn)+O(n)=O(nlogn)

16.3-2

We assume by contradiction that a non-full binary tree can correspond to some optimal prefix code.

In this case there must be a node v having a single child node u.

We distinguish two cases:

1). v is the root of the tree; then, we can remove it and use its child u as root

2). v is not the root of the tree => it must have a parent node z. In this case, we remove v and u becomes the child of node z.

In both situations, the necessary no. of bits for encoding any leaf of the subtree has decreased. On the other hand, the rest of the tree was not affected by anything. In this new tree, the average bit length is smaller than the previous, which contradicts the fact the original non-full tree could have been optimal.

Therefore, it is not true that a non-full binary tree can correspond to an optimal prefix code. So, if the tree corresponds to some optimal prefix code, then it must be a full binary tree.

16.3-6

From the previous problem, we know the associated binary tree (i.e. for this optimal prefix code) must be full. Or, a full binary tree has n leaf nodes and n-1 internal nodes for a total of n+(n-1)=2n-1 nodes.

The set C has n members => [logn] bits suffice to represent one of them. (here [] represents the ceiling).

Therefore, n[logn] bits suffice representing all n members of C.

We can use the preorder traversal of the uniquely determined full binary tree and encode by a 0 the start of every leaf and by a 1 the start of any internal node (or vice versa). Since the tree is unique, there is no ambiguity which can arise during the traversal. In conclusion, we have used all 2n-1+n[logn] bits allowed by the hypothesis of the problem.

16.3-7

In essence, if we group at every step the 3 symbols having the smallest frequencies (and repeat the process until the end), nothing changes in the analysis done in the "classic" binary case (at high or low level details). It is trivial noticing that a ternary structure results (i.e. one where we use 0,1, and 2 to traverse the tree, as an extension from the use of 0 and 1 only like in the traditional case). In addition, the obtained tree is optimal due to the same reasons as those listed in the analysis of traditional Huffman binary encoding. The only problem which might arise is when the no. of nodes is even because a full ternary tree cannot be obtained otherwise. However, in those cases, it is possible to insert "dummy" nodes (i.e. frequency =0).