CS21003 ALGORITHMS-1
(Solution : Greedy Strategy)
Date: October 17, 2020

Question 1:

Put as many songs (from song 1 to song g) on the 1st CD as possible without exceeding the m minute limit. Then iteratively repeat this procedure for the remaining CDs. Since this just requires keeping a running sum in which each of the n song lengths are included once, clearly the above is an O(n) algorithm.

Question 2:

- input: (a1,d1), (a2,d2), ...
- Convert input in individual event pair : (time,arrival/departure)
- Sort all the events pair based on time (first element of pair)
- max = 0
- currentGuest = 0
- For all events pairs do
  - If event[second] element == arrival then currentGuest++
  - else currentGuest–
  - If currentGuest > max then max = currentGuest
- Return max

Time Complexity: O(nlogn)

Question 3:

we can defeat the i^{th} wight with Drogon (D), Rhaegal (R) or Viserion (V).
so we have the following equations:
cost[i,D] = min(cost[i-1,R], cost[i-1,V]) + cost of defeating i with D.
cost[i,R] = min(cost[i-1,D], cost[i-1,V]) + cost of defeating i with R.
cost[i,V] = min(cost[i-1,D], cost[i-1,R]) + cost of defeating i with V.
Final Min Cost = min (cost[n,D], cost[n,R], cost[n,V])
Time Complexity : O(n)

Question 4:

Q3. Here is a greedy algorithm for this problem. Start at the western end of the road and begin moving east until the first moment when there's a house h exactly four miles to the west. We place a base station at this point (if we went any farther east without placing a base station, we wouldn't cover h). We then delete all the houses covered by this base station, and iterate this process on the remaining houses.

Here's another way to view this algorithm. For any point on the road, define its *position* to be the number of miles it is from the western end. We place the first base station at the easternmost (i.e. largest) position s_1 with the property that all houses between 0 and s_1 will be covered by s_1. In general, having placed $\{s_1, \ldots, s_i\}$, we place base station $i+1$ at the largest position s_{i+1} with the property that all houses between s_i and s_{i+1} will be covered by s_i and s_{i+1}.

Let $S = \{s_1, \ldots, s_k\}$ denote the full set of base station positions that our greedy algorithm places, and let $T = \{t_1, \ldots, t_m\}$ denote the set of base station positions in an optimal solution, sorted in increasing order (i.e. from west to east). We must show that $k = m$.

We do this by showing a sense in which our greedy solution S "stays ahead" of the optimal solution T. Specifically, we claim that $s_i \geq t_i$ for each i, and prove this by induction. The claim is true for $i = 1$, since we go as far as possible to the east before placing the first base station. Assume now it is true for some value $i \geq 1$, this means that our algorithm's first i centers $\{s_1, \ldots, s_i\}$ cover all the houses covered by the first i centers $\{t_1, \ldots, t_i\}$. As a result, if we add t_{i+1} to $\{s_1, \ldots, s_i\}$, we will not leave any house between s_i and t_{i+1} uncovered. But the $(i+1)^{\text{st}}$ step of the greedy algorithm chooses s_{i+1} to be *as large as possible* subject to the condition of covering all houses between s_i and s_{i+1}; so we have $s_{i+1} \geq t_{i+1}$. This proves the claim by induction.

Finally, if $k > m$, then $\{s_1, \ldots, s_m\}$ fails to cover all houses. But $s_m \geq t_m$, and so $\{t_1, \ldots, t_m\} = T$ also fails to cover all houses, a contradiction.

Question 5:

Q4. An optimal algorithm is to schedule the jobs in decreasing order of w_i/t_i. We prove the optimality of this algorithm by an exchange argument.

Thus, consider any other schedule. As is standard in exchange arguments, we observe that this schedule must contain an *inversion* — a pair of jobs i, j for which i comes before j in the alternate solution, and j comes before i in the greedy solution. Further, in fact, there must be an adjacent such pair i, j. Note that for this pair, we have $w_j/t_j \geq w_i/t_i$, by the definition of the greedy schedule. If we can show that swapping this pair i, j does not increase the weighted sum of completion times, then we can iteratively do this until there are no more inversions, arriving at the greedy schedule without having increased the function we're trying to minimize. It will then follow that the greedy algorithm is optimal.

So consider the effect of swapping i and j. The completion times of all other jobs remain the same. Suppose the completion time of the job before i and j is C. Then before the swap, the contribution of i and j to the total sum was $w_i(C + t_i) + w_j(C + t_i + t_j)$, while after the sum it is $w_j(C + t_j) + w_i(C + t_i + t_j)$. The difference between the value after the swap, compared to the value before the swap is (canceling terms in common between the two expressions) $w_i t_j - w_j t_i$. Since $w_j/t_j \geq w_i/t_i$, this difference is bounded above by 0, and so the total weighted sum of completion times does not increase due to the swap, as desired.

Question 6:

Q5. Let $I_1, \ldots, I_n$ denote the n intervals. We say that an *I_j-restricted solution* is one that contains the interval I_j.

Here is an algorithm, for fixed j, to compute an I_j-restricted solution of maximum size. Let x be a point contained in I_j. First delete I_j and all intervals that overlap it. The remaining intervals do not contain the point x, so we can "cut" the time-line at x and produce an instance of the Interval Scheduling Problem from class. We solve this in $O(n)$ time, assuming that the intervals are ordered by ending time.

Now, the algorithm for the full problem is to compute an I_j-restricted solution of maximum size for each $j = 1, \ldots, n$. This takes a total time of $O(n^2)$. We then pick the largest of these solutions, and claim that it is an optimal solution. To see this, consider the optimal solution to the full problem, consisting of a set of intervals S. Since $n > 0$, there is some interval $I_j \in S$; but then S is an optimal I_j-restricted solution, and so our algorithm will produce a solution at least as large as S.

Question 7:

Q1. Say n boxes arrive in the order $b_1, \ldots, b_n$. Say each box b_i has a positive weight w_i, and the maximum weight each truck can carry is W. To pack the boxes into N trucks *preserving the order* is to assign each box to one of the trucks $1, \ldots, N$ so that:

- No truck is overloaded: the total weight of all boxes in each truck is less or equal to W.
- The order of arrivals is preserved: if the box b_i is sent before the box b_j (i.e. box b_i is assigned to truck x, box b_j is assigned to truck y, and $x < y$) then it must be the case that b_i has arrived to the company earlier than b_j (i.e. $i < j$).

We prove that the greedy algorithm uses the fewest possible trucks by showing that it "stays ahead" of any other solution. Specifically, we consider any other solution and show the following. If the greedy algorithm fits boxes $b_1, b_2, \ldots, b_j$ into the first k trucks, and the other solution fits $b_1, \ldots, b_i$ into the first k trucks, then $i \leq j$. Note that this implies the optimality of the greedy algorithm, by setting k to be the number of trucks used by the greedy algorithm.

We will prove this claim by induction on k. The case $k - 1$ is clear; the greedy algorithm fits as many boxes as possible into the first truck. Now, assuming it holds for $k - 1$: the greedy algorithm fits j' boxes into the first $k - 1$, and the other solution fits $i' < j'$. Now, for the k^{th} truck, the alternate solution packs in $b_{i'+1}, \ldots, b_i$. Thus, since $j' \geq i'$, the greedy algorithm is able at least to fit all the boxes $b_{j'+1}, \ldots, b_i$ into the k^{th} truck, and it can potentially fit more. This completes the induction step, the proof of the claim, and hence the proof of optimality of the greedy algorithm.