The last two problems of this assignment covers greedy algorithms, which we will begin to cover on next Wednesday. So you could wait till Wednesday to work on these two problems.

1. The well-known mathematician George Polya posed the following false "proof" showing through mathematical induction that actually, all horses are of the same color.

Base case: If there’s only one horse, there’s only one color, so of course its the same color as itself.

Inductive case: Suppose within any set of \( n \) horses, there is only one color. Now look at any set of \( n+1 \) horses. Number them: 1, 2, 3, ..., \( n \), \( n+1 \). Consider the sets \{1, 2, 3, ..., \( n \}\} and \{2, 3, 4, ..., \( n+1 \}\}. Each is a set of only \( n \) horses, therefore within each there is only one color. But the two sets overlap, so there must be only one color among all \( n+1 \) horses.

Identify what is wrong with this proof.

For the case of 2 horses, the two subsets do not overlap. So we could not go from \( n=1 \) to \( n=2 \).

2. Given two sorted arrays \( a[1, ..., n] \) and \( b[1, ..., n] \), given an \( O(\log n) \) algorithm to find the median of their combined \( 2n \) elements. (Hint: use divide and conquer).

Solution:

For the base case, if \( n \) is small, say 1 or 2. We can just directly compute the combined median. For larger \( n \), the algorithm works by comparing the medians of the two array (the median of sorted arrays can be computed in constant time). If the two medians are the same, then the combined median is the same value, because exactly 50% of \( a \) and 50% \( b \) are smaller than this median. If \( a \)'s median is greater than \( b \)'s median, the combined median must be either in the first half of \( a \) or the second half of \( b \). So it can be found by recursively finding the combined median for \( a_L \) and \( b_R \).

In contrast, if \( a \)'s median is smaller than \( b \)'s, the combined median must be either in the second half of \( a \) or the first half of \( b \) and can be found by recursively finding the combined median of \( a_R \) and \( b_L \). This algorithm will have one recursive call on a half-sized subproblem and constant time non-recursive computation. That is \( T(n) = T(n/2) + c = \Theta(\log n) \).
3. Given an array \( A \) of \( n \) distinct numbers whose values \( A[1], A[2], \ldots, A[n] \) is unimodal: that is, for some index \( p \in [1, n] \), the values in the array first increases up to position \( p \), then decrease the remainder of the way. For example \([1, 2, 5, 9, 7, 3]\) is one such array with \( p = 4 \). Please design an \( O(\log n) \) algorithm to find the peak \( p \) given such an array \( A \).

**Solution:** Since we are looking for \( \log n \) run time, our solution will likely resemble binary search, and each recursive call we should shrink the problem size by a constant factor. In particular, The base case is when \( A \) has only one or two element, we can directly find the peak and return the index. For \( n > 2 \), we can focus on \( A[n/2] \) and its two neighboring elements. If they display an increasing trend, we know the peak must be in the right half of \( A \) and find it by recursively finding the peak in \( A_R \). If they display an decreasing trend, we know the peak must be in the left half of \( A \) and find it by recursively finding the peak in \( A_L \). If \( A[n/2] \) is bigger than its two neighbors, we know \( n/2 \) is the peak. The run time of this algorithm is \( T(n) = T(n/2) + c = \Theta(\log n) \).

4. **Interval scheduling.** We are given a set of requests for using a resource. Each request \( i \) specifies a starting time \( s(i) \) and an end time \( f(i) \). The resource can only accommodate one request at a time. If two requests overlap in time, they are incompatible and cannot be both fulfilled. The goal is to identify a maximum subset of compatible requests. One possible greedy strategy is to select at each step the request that is compatible with the maximum number of the remaining requests. Will this greedy strategy lead to an optimal solution? If so, provide a proof. If not, provide a counter example.

**solution:** This strategy is not optimal. See the top example for a case where this strategy will lead to suboptimal solution.

5. You and your friends are taking a long hiking trip of \( L \) miles, along which there are \( n \) camping sites located at distances \( x_1, x_2, \ldots, x_n \) respectively from the start of the trip. You can hike at most \( d \) miles per day, by the end of which you must stop and camp for the night. You need make a valid trip plan that takes the minimum number of camping stops. The plan should specify which camping sites to use, and it is only **valid** if any two consecutive stops are no more than \( d \) miles apart. Your friend
proposed the following strategy: each time you come to a camp site, check whether you can make it to the next site before the end of the day (i.e., before finishing the d miles quota for the day. We assume this can always be determined correctly), if so, keep hiking. If not, stop for the night.

This is in fact a greedy algorithm, which simply chooses to hike as long as possible each day. Prove that this greedy algorithm achieves the optimal solution, i.e., it uses the minimum number of stops. (Hint: construct a proof that is similar to the interval scheduling proof, which shows that the greedy algorithm stays ahead.)

**Solution:** Let \( A = \{i_1, i_2, ..., i_k\} \) be the set of stopping points selected by the greedy algorithm. Let \( O = \{j_1, j_2, ..., j_m\} \) be a set of optimal stopping points that is smaller (i.e, \( m < k \)). Assume both are ordered based on the distances from the starting point.

First, we argue the following statement, which essentially establishes that the greedy algorithm stays ahead (travels more distances) than the optimal solution.

*For each \( l = 1, ..., m \), we have \( x_{i_l} \geq x_{j_l} \).

To prove this, we can use induction on \( l \).

**Inductive assumption:** assume that the statement is true for \( 1 \leq k < l \).

**Base case:** For \( l = 1 \), according to the algorithm, it will select the furthest site that can be reached in one day. Thus the statement is true trivially.

**Inductive case:** For \( l > 1 \), since \( 1 \leq l - 1 < l \), the inductive assumption implies that \( x_{i_{l-1}} \geq x_{j_{l-1}} \). We would use this to prove that \( x_{i_l} \geq x_{j_l} \).

Now let’s prove it by contradiction.

We now assume the opposite is true, i.e., \( x_{i_l} < x_{j_l} \). Because the algorithm operates by going as far as possible for each day and \( x_{i_l} < x_{j_l} \), \( j_l \) must not be reachable from site \( i_{l-1} \) in one day, otherwise the algorithm would have selected \( j_l \) as its camp site for that day. Since \( x_{i_{l-1}} \geq x_{j_{l-1}} \), it follows that the distance between \( j_{l-1} \) and \( j_l \) must be greater than \( d \), this contradicts with the fact that \( O \) is a valid plan. Thus we must have \( x_{i_l} \geq x_{j_l} \).

Now since the end of the trail can be reached within one day from \( j_m \), the above property allows to conclude that it can be reached within one day from \( i_m \) as well. Thus we must have \( k = m \). This completes the proof.