Homework 3

1. Consider a collection of n ropes which have lengths $L_1, L_2, \ldots, L_n$, respectively. Two ropes of length L and L' can be connected to form a single rope of length $L + L'$, and doing so has a cost of $L + L'$. We want to connect the ropes, two at a time, until all ropes are connected to form one long rope. Design an efficient algorithm for finding an order in which to connect all the ropes with minimum total cost. You do not need to prove that your algorithm is correct. (20 points)

   Enter all rope segments into a min-heap with the length of the rope segment being its key value. Iteratively pop the 2 shortest ropes and connect them. Then insert the new resulting rope segment (with the key value being the sum of the lengths of the ropes that are connected together) back into the heap. Continue until you are left with only 1 rope segment in the heap.

2. Suppose you want to drive from USC to Santa Monica. Your gas tank, when full, holds enough gas to drive p miles. Suppose there are n gas stations along the route at distances $d_1 \leq d_2 \leq \cdots \leq d_n$ from USC. Assume that the distance between any neighboring gas stations, and the distance between USC and the first gas station, as well as the distance between the last gas station and Santa Monica, are all at most p miles. Assume you start from USC with the tank full. Your goal is to make as few gas stops as possible along the way. Design an efficient algorithm for determining the minimum number of gas stations you must stop at to drive from USC to Santa Monica. Prove that your algorithm is correct. Analyze the time complexity of your algorithm. (25 points)

   Greedy algorithm: The greedy strategy we adopt is to go as far as possible before stopping for gas. That is, when you are at the i-th gas station, if you have enough gas to go to the $(i+1)$-th gas station, then skip the i-th gas station. Otherwise stop at the i-th station and fill up the tank.

   Proof of optimality:
   The proof is similar to that for the interval scheduling solution we did in lecture. We first show (using mathematical induction) that our gas stations are always to the right of (or not to the left of) the corresponding base stations in any optimal solution. Using this fact, we can then easily show that our solution is optimal using proof by induction.

(a) First we show our gas stations are never 'earlier' than the corresponding gas station in any optimal solution:
Let $g_1, g_2, \ldots, g_m$ be the set of gas stations at which our algorithm made us refuel. Let $h_1, h_2, \ldots, h_k$ be an optimal solution. We first prove that for any indices $i < m$, $h_i \leq g_i$.

Base case: Since it is not possible to get to the (g_1+1)-th gas station without stopping, any solution should stop at either g_1 or a gas station before g_1, thus $h_1 \leq g_1$.

Induction hypothesis: Assume that for the greedy strategy taken by our algorithm, $h_c \leq g_c$.

Inductive step: We want to show that $h_c+1 \leq g_c+1$. It follows from the same reasoning as above. If we start from h_c, we first get to g_c (IH) and, when leaving g_c, we now have at most as much fuel as we did if we had refilled at g_c. Since it is not possible to get to $g_{(c+1)+1}$ without any stopping, any solution should stop at either g_{c+1} or a gas station before g_{c+1} , thus $h_{c+1} \leq g_{c+1}$

(b) Now assume that our solution requires m gas stations and the optimal solution requires fewer gas stations. We now look at our last gas station. The reason we needed this gas station in our solution was that there is a point on I-10 after this gas station that cannot be reached with the amount of gas when we left gas station m-1. Therefore we would not have enough gas if we left gas station m-1 in any optimal solution. Therefore, any optimal solution also would require another gas station.

The running time is O(n) since we at most make one computation/decision at each gas station.

3. Suppose you are given two sequences A and B of n positive integers. Let a_i be the i-th number in A, and let b_i be the i-th number in B. You are allowed to permute the numbers in A and B (rearrange the numbers within each sequence, but not swap numbers between sequences), then you receive a score of $\prod_{1 \leq i \leq n} a_i^{b_i}$. Design an efficient algorithm for permuting the numbers to maximize the resulting score. Prove that your algorithm maximizes the score and analyze your algorithm's running time (25 points).

   Algorithm: Sort A and B in increasing order. This results in a maximum score.

Without loss of generality, assume A is sorted in increasing order, since otherwise, the permutation that sorts A can be applied to A and B to yield a solution with the same score, but with A sorted.

Consider an arbitrary solution where B is not sorted. Then there exists an inversion / a pair of indices i and j where $b_i > b_j$ but $i < j$. Since A is sorted, we know $a_i \leq a_j$. Then the ratio between the score achieved when b_i and b_j are swapped versus the current choice of B is

$(a_i^{b_j} \times a_j^{b_i}) / a_i^{b_i} \times a_j^{b_j} = (a_j / a_i)^{b_i - b_j} \geq 1$

The inequality is due to the fact that $a_i \leq a_j$, so $(a_j / a_i) \geq 1$, and $b_i > b_j$, so $b_i - b_j > 0$ (a number that is ≥ 1 and taken to a positive power is ≥ 1). When b_i and b_j are swapped, the score is multiplied by this ratio, which, since the ratio is ≥ 1, increases the score. By iteratively removing inversions this way, we see that the solution where B is sorted achieves an equal or greater score compared to the solution where B is not sorted. Therefore we conclude that the solution where B is sorted is optimal.

Runtime Complexity : O(n log n) for sorting each of the sets.

4. The United States Commission of Southern California Universities (USC-SCU) is researching the impact of class rank on student performance. For this research, they want to find a list of students ordered by GPA containing every student in California. However, each school only has an ordered list of its own students sorted by GPA and the commission needs an algorithm to combine all the lists. Design an efficient algorithm with running time $O(m \log n)$ for combining the lists, where m is the total number of students across all colleges and n is the number of colleges. (20 points)

To construct a sorted list L containing all students across all colleges, first construct a min-heap H initialized with the lowest-GPA student from each college. Iteratively pop the lowest-GPA student x in H with ExtractMin and append it to the end of L, then push the student with the next highest GPA in x's college (unless all the students from that college are already in L).

Since the heap H initially contains n students, the runtime complexity of initializing the heap is $O(n \log n) = O(m \log n)$. Since there are at most n students in the heap H at any time, each pop and push heap operation takes $O(\log n)$ time. Since we perform m pop and push operations, the runtime complexity of the algorithm is $O(m \log n)$.

5. The array A below holds a max-heap. What will be the order of elements in array A after a new entry with value 19 is inserted into this heap? Show all your work.
   A = {16, 14, 10, 8, 7, 9, 3, 2, 4, 1} (10 points)

Initial Array

16 14 10 8 7 9 3 2 4 1

Array after inserting 19

16 14 10 8 7 9 3 2 4 1 19

19 is greater than 7(the element at index 11/2=5), so swap

16 14 10 8 19 9 3 2 4 1 7

19 is greater than 14(the element at index 5/2=2), so swap

16 19 10 8 14 9 3 2 4 1 7

19 is greater than 16(the element at index 2/2=1), so swap

Element 19 16 10 8 14 9 3 2 4 1 7

Final Array = {19, 16, 10, 8, 14, 9, 3 , 2, 4, 1, 7}