HW3

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)

Step 1: Put the lengths of all ropes into a min-heap with the rope length as the key value $\Rightarrow O(n)$

Step 2: Take the two shortest ropes from the pile at a time and connect them. $\Rightarrow O(\log n)$

Step 3: Put the newly generated rope back to the heap. The new key value will be the sum of the shortest previous two ropes. $\Rightarrow O(\log n)$

Step 4: Repeat previous process until only one segment retains in the pile. $(n-1)\, O(\log n)$

$\Rightarrow$ Overall $O(n \log n)$

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)

Algorithm: Greedy. We will maximize the distance traveled before needing to refuel. Only if the fuel is inadequate to reach the next station, a stop at the ith station for refueling is made. Otherwise, we will go to (i+1)th station directly. Stops are made only when absolutely necessary, optimizing the journey by minimizing fuel stops.

Proof by mathematical induction

Suppose A is our solution, and O is the optimal solution O.

A A_1 A_2 ... A_k A_{k+1}

O O_1 O_2 ... O_k O_{k+1}

Our gas station stops are always to the right of the corresponding gas station stops in any optimal solution.

Base case: we stop at A_1, since we could not get to A_2 without refueling. Thus, for any optimal solution, we have to stop at/before A_1., $\Rightarrow O_1 \leq A_1$

Inductive Step: Assume our kth gas station stop is to the right of the kth gas station stop in optimal solution

$\Rightarrow O_k \leq A_k$

We can now show that our k+1st gas station stop is also to the right of k+1st gas station stop in the optimal solution.

To prove this, we know when we stop at O_k to get refueled. we then get to A_k, we can choose stop or not stop. If stopped, then the gas is refueled and we can make it to A_{k+1}. If not stopped, then it is impossible to get to A_{k+1} without stopping between O_k and A_{k+1}, thus we need to stop at O_{k+1} which is before A_{k+1}, $\Rightarrow O_{k+1} \leq A_{k+1}$

Proof by contradiction:

Now, assume that our solution required n gas station stops and the optimal solution requires fewer gas station stops. We now look at our last stop. the reason we needed this stop in our solution was that we couldn't get to santa monica with the full gas fueling at $(n-1)$ gas station stop. Then, we would not have enough gas if we stopped at gas station $n-1$ in optimal solution. (since our gas station stop are always to the right of the corresponding gas station stops in the optimal solution). So, the optimal solution will also need another gas station stop to refuel.

Complexity of Solution will be $O(n)$ since we need to make decision at each station (stop or not.

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** $\prod_{1 \le i \le n} a_i^{b_i}$. 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).

$\prod_{1 \le i \le n} a_i^{b_i}$

Algorithm: Rearrange the numbers within each sequence so that A and B are sorted in the same ascending order. Then we can maximize the result by pairing the corresponding ith number in both A and B. Runtime Complexity is $O(n \log n)$ for sorting in ascending order.

Proof by contradiction:

Solution S: Assume the ascending order of A is: i, j, k, m

$a_1, a_2, a_3, a_4, \ldots, a_i, \ldots, a_j, \ldots, a_n$

The ascending order of B is

$b_1, b_2, b_3, b_4, \ldots, b_i, \ldots, b_j, \ldots b_n$.

Assume there is an optimal solution. We got:

Solution O: A: $a_1, a_2, a_3, a_4, \ldots a_i, \ldots a_j, \ldots a_n$ where $i < j$

B: $b_1, b_2, b_3, b_4 \ldots b_j, \ldots b_i, \ldots b_n$

In other words, the optimal solution will pair a_i and b_j up. a_j and b_i up, but keep other pairs same as in solution S.

$$\frac{\text{ResultingScore}(S)}{\text{ResultingScore}(O)} = \frac{a_1^{b_1} \times a_2^{b_2} \times \cdots \times a_i^{b_i} \times \cdots \times a_j^{b_j} \times a_n^{b_n}}{a_1^{b_1} \times a_2^{b_2} \times \cdots \times a_i^{b_j} \times \cdots \times a_j^{b_i} \times a_n^{b_n}}$$

$$= \frac{a_i^{b_i} a_j^{b_j}}{a_i^{b_j} a_j^{b_i}}$$

$$= \left(\frac{a_i}{a_j}\right)^{b_i} \left(\frac{a_j}{a_i}\right)^{b_j} = \left(\frac{a_i}{a_j}\right)^{(b_i - b_j)}$$

∵ $i < j$

∴ $a_i < a_j$, $b_i < b_j$ in S

∴ $0 < \frac{a_i}{a_j} < 1$, $b_i - b_j < 0$

∴ $\left(\frac{a_i}{a_j}\right)^{b_i - b_j} > 1$

∴ ResultingScore(S) > ResultingScore(O)

This is a contradiction of O is an optimal solution.

Thus, the optimal solution should be ordering the A and B in same ascending way.

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)

We assume GPA is sorted in ascending order.

Step 1: Initialize a minimum heap H, which can store student information of all schools. (size n)

Iterate through the student lists of each university and add the first one of each list to the heap $\Rightarrow O(n)$

Step 2: Create pointer for each element in the heap H, which points to the second element in its sorted school list.

$\Rightarrow O(n)$

Step 3: Remove the smallest (the first) element S in heap H.

$\Rightarrow O(\log n)$

Add element S to the final combined list L.

$\Rightarrow O(1)$

Find out which school this minimal element comes from which stored with the student info. From the list where student is located, take the next element E pointed by the previous removed element, then insert it to the heap H.

$\Rightarrow O(\log n)$

Update the pointer so that it points to next element of E

Step 4. Repeat the above process until all students (size m) have been processed. Each time the minimum element is taken out and the new element is inserted ($\log n$). the minimum heap is updated and its structure is maintained. So that current minimum element can be quickly taken on next time.

$\Rightarrow$ Overall $O(m \log n)$

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)