Algorithms Homework #2

Due Mar 31, 2022 before class.

Collaboration policy: You can discuss the problem with other students, but you must obtain and write the final solution by yourself. Please specify all of your collaborators (name and student id) for each question. If you solve some problems by yourself, please also specify "no collaborators". Homeworks without collaborator specification will not be graded.

Problem 1 (10%)

Answer the following questions for Bubble Sort, Merge Sort, Insertion Sort, Quicksort (deterministic version) and Heap sort. No Explanation is needed.

1. What is the worst-case time complexity of the algorithm?
2. Is it stable?
3. How much time does it take to sort sequences that are already sorted? (answer using $\Theta()$ notation)
4. How much time does it take to sort sequences that are reversed? (answer using $\Theta()$ notation)

Problem 2 (15%)

Given n distinct real numbers $a_1, a_2, \ldots, a_n$ (not sorted). $a_1 = 0$ and $a_2 = n$. Your goal is to find a pair of numbers a_i and a_j such that $|a_i - a_j| > 1$, but there is no other a_k with value in between. Design an $O(n)$-time algorithm which finds one such pairs. Briefly justify the correctness and the running time of the algorithm. (For example, if the input is $0, 4, 4.2, 1.3$, the algorithm may output either (0, 1.3) or (1.3, 4).) You may assume that n is a power of 2.

Problem 3 (30%)

An array of n elements is almost sorted if and only if every element is at most k spots away from its actual location. Assuming that you can only perform pairwise comparisons,

1. Design a $O(n \log k)$ algorithm to sort almost sorted arrays. Briefly justify the correctness and the running time.
2. Formally prove that any algorithm which can sort almost sorted arrays must have running time $\Omega(n \log k)$, You may assume that n is a multiple of k.

In problems 4 and 5, we consider a variation of the meeting selection problem discussed in class. One day, Prof. Chen has n meetings $M_1, M_2, \ldots, M_n$ on his schedule. Meeting M_i starts at time s_i and ends at time t_i for every i. He wants to select a set of meetings to attend, but if he choose to attend a meeting, he is not allowed to arrive late or leave early.

Problem 4 (15%) In this problem, Prof. Chen tries to maximize the number of meetings he can attend. This is exactly the problem we talked about in class. Assume that the optimal solution chooses k meetings. Consider the greedy problem which always selects the meeting with the least number of conflicts. Prove that this algorithm always chooses at least $\lceil \frac{k}{2} \rceil$ meetings or provide a counter example.

Problem 5 In this problem, Prof. Chen wants to maximize the total meeting time (instead of the number of meetings). Also, assume that $t_1 \leq t_2 \leq \cdots \leq t_n$. Solve the following two problems. (The additional assumption in 5-1 does not apply in 5-2 and vice versa.)

1. (10%) Given an additional assumption that Prof. Chen must attend at least k meetings. Design an algorithm to find the optimal set of meetings (which maximizes the total meeting time). The running time of your algorithm must be polynomial in nk. Briefly justify the correctness and analyze the running time.

2. (10%) Given an additional assumption: For every i, M_i and M_{i+2} cannot both be attended, even if the meeting time does not conflict each other. Design an algorithm to find the maximum total meeting time (you do not need to find the optimal set of meetings). The running time must be polynomial in n. Briefly justify the correctness and analyze the running time.

Problem 6 (10%)

A skillful baker can make m types of breads $b_1, b_2, \ldots, b_m$. Making bread b_i generates profit p_i. Given that

1. The baker has enough ingredient to make any bread once.

2. He does not have enough ingredient to make any bread twice.

3. For every i, breads b_i and $b_{\lfloor \frac{i}{2} \rfloor}$ share a common ingredient and cannot be both made. (ex: b_1 and b_2 cannot both be made, but b_1 and b_4 can.)

Design an algorithm to find the set of breads which the baker can make to maximize his total profit. Any algorithm that runs in time polynomial in m suffices. Briefly explain the correctness and analyze the running time.