Algorithms Homework #2

Due Nov 2, 2020 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 (10%)

Cyclic shift is a permutation which shifts all elements of an array by a fixed number of positions (wrapped around). For example: the sequence $1, 2, 3, 4, 5$ becomes $5, 1, 2, 3, 4$ when cyclic-shifted by one position, and becomes $3, 4, 5, 1, 2$ when cyclic-shifted by three positions. Given an array of n distinct elements $A[1 \dots n]$ which is a cyclic shift of a sorted array. Design an $O(\log n)$-time algorithm to find the minimum. Briefly justify the correctness and the running time. Any algorithm which requires $\Omega(n)$ running time will receive very little credit.

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, a company wants to construct machines to produce surgical masks for its own workers. The requirement is n masks per day. There are m types of machines $T_1, T_2, \dots, T_m$. Machine T_i costs c_i and can produce n_i masks per day. Due to insufficient supply, only one machine of each type can be made. The goal is to satisfy the requirement (n per day) while minimizing the total cost.

For simplicity, assume that the machine types are sorted such that the price per mask is in the increasing order. In other words, $\frac{c_1}{n_1} \leq \frac{c_2}{n_2} \leq \dots \frac{c_m}{n_m}$. Also, assume that $n_i \leq n$ for all i.

Problem 4 In this problem, consider the greedy algorithm which constructs $T_1, T_2, \dots$ until there is enough supply.

1. (10%) Find an example in which the algorithm output is at least ten times more expensive than the optimal solution. If you cannot manage this problem, find an example in which the algorithm does not find the optimal solution gives partial credits.
2. (10%) Given an extra assumption that all c_is are within a factor of four to each other (i.e. $c_i \leq 4c_j$ and $c_j \leq 4c_i$ for every i, j). Prove that the algorithm output is at most five times more expensive than the optimal solution.

Problem 5

1. (10%) Design an algorithm to find the optimal solution (cheapest set of machines which satisfy the requirement). Your algorithm must run in time $O(mn)$ and use $O(mn)$ memory to receive full credits. If you cannot manage this problem, any algorithm which computes the minimum cost in polynomial time gives partial credits. Briefly justify the correctness and analyze the running time.
2. (10%) Given an additional assumption: For every integer i, if machine T_{2i-1} is made, then machines T_{2i} and T_{2i+1} cannot be made. (Machines T_{2i} and T_{2i+1} do not conflict. Machines T_{2i-1} and T_{2i+3} do not conflict.) Design a polynomial time algorithm which finds the minimum cost possible. Briefly justify the correctness and analyze the running time. You may assume that m is an even number.

Problem 6 (10%)

A skillful baker can make m types of breads $b_1, b_2, \dots, b_m$. 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.