Homework 2

Course: Algorithm Design and Analysis **Semester**: Spring 2024

Instructor: Shi Li **Due Date**: **2024/4/7**

Student Name: _____ Student ID: _____

Problems	1	2	3	4	5	6	Total
Max. Score	15	15	20	15	15	20	100
Your Score							

Remarks:

- In all the algorithm design problems, a correct pseudo-code or description of the algorithm will give you a majority of the points.
- If the algorithm you designed is a greedy algorithm, you need to prove its correctness to get a full score. The most convenient way to do this is to describe the three elements: the decision you make for each step, proof of the safety of the decision, a reduction of the instance after you made the decision. If additionally, you are asked to achieve a certain running time, a pseudo-code on top of the three elements is sufficient.
- If the algorithm you designed is a divide and conquer algorithm, you need to describe the input and the output of the recursion, and the recurrence for the running time, and give the final running time.
- The examples are given to you to help you understand the problems. There might be multiple optimum solutions for the given input instance; it is OK if your algorithm outputs a different optimum solution from the one provided.

Problem 1. In the interval covering problem, we are given n intervals $[s_1, t_1), [s_2, t_2), \cdots, [s_n, t_n)$ such that $\bigcup_{i\in[n]}[s_i, t_i) = [0, T)$ for some given T. The goal of the problem is to return a smallest-size set $S \subseteq [n]$ such that $\bigcup_{i\in S}[s_i, t_i) = [0, T)$. Design a greedy algorithm to solve this problem. To get a full mark for this problem, the running time of the algorithm should be $O(n \log n)$.

For example, consider the instance in Figure 1. You can use intervals $1, 4, 7$ to cover the whole interval $[0, 160)$, and the set $\{1, 4, 7\}$ is the optimum solution for the instance.

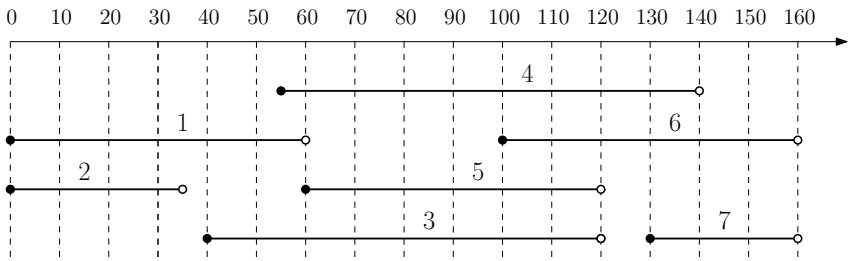

Figure 1: Instance for Interval Covering.

Problem 2. Given two sequences A and B of letters, the sequence A is a sub-sequence of B, if A can be obtained from B by removing some letters. For example, "abcadb" is a sub-sequence of "badbaccaddfb" as you can remove letters as follows: "~~bad~~b~~ac~~ca~~ddf~~b".

Given two sequences A and B of letters with total length n, design an $O(n)$-time algorithm to decide if A is a sub-sequence of B.

Problem 3. You are given n jobs which needs to be processed on one machine. Each job j has an arrive time r_j, a deadline d_j and a processing time p_j, where $r_j \in \mathbb{Z}_{\geq 0}$ and $p_j, d_j \in \mathbb{Z}_{>0}$. A job j can only be processed during the time interval $(r_j, d_j]$. The processing of jobs can be interrupted and resumed later, and a job j is completed if it is processed for p_j units of time during the interval $(r_j, d_j]$. Your goal is to check if all the jobs can be completed. Design an $O(n \log n)$ time algorithm to solve the problem.

Formally, the time is slotted into unit time slots $1, 2, 3, \cdots$. During any time slot t, you can choose to process a job j on the time slot, if $r_j < t \leq d_j$. You can process at most one job in a time slot. The jobs can be completed if every job j can be processed during p_j time slots.

For example, consider the instance with 3 jobs given by Table 1. The 3 jobs can all be completed: A feasible solution is given in Figure 2. So, for the instance you should output "yes". However, if p_c is 2 instead of 1, then you should output "no".

job indices	a	b	c
r_j	0	3	7
d_j	5	10	9
p_j	4	5	1

Table 1: Instance for Problem 3.

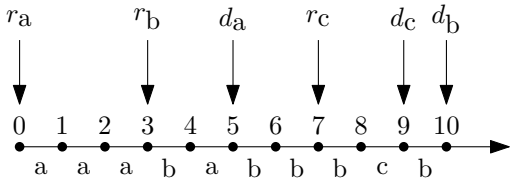

Figure 2: A feasible schedule of the instance.

Problem 4. We consider the following problem of counting significant inversions. Given an array A of n positive integers, a pair $i, j \in \{1, 2, 3, \cdots, n\}$ of indices is called a significant inversion if $i < j$ and $A[i] > 2A[j]$. The goal of the problem is to count the number of significant inversions for a given array A. Give a divide-and-conquer algorithm that runs in $O(n \lg n)$ time to solve the problem.

Problem 5. Given an array $A[1 \,..\, n]$ of n **distinct** numbers, we say that some index $i \in \{1, 2, 3 \cdots, n\}$ is a local minimum of A, if $A[i] < A[i-1]$ and $A[i] < A[i+1]$ (we assume that $A[0] = A[n+1] = \infty$). Suppose the array A is already stored in memory. Give an $O(\log n)$-time algorithm to find a local minimum of A.

For example, if the array is $A[1..10] = (50, 10, 20, 40, 100, 70, 30, 80, 90, 60)$, your algorithm can return $2, 7$ or 10. Notice that it only needs to return one of them.

Problem 6. (Exercise 4 on Page 247 of KT book) You've been working with some physicists who need to study, as part of their experimental design, the interactions among large numbers of very small charged particles. Basically, their setup works as follows. They have an inert lattice structure, and they use this for placing charged particles at regular spacing along a straight line. Thus we can model their structure $1, 2, 3, \cdots, n$ on

the real line; and at each of these points j, they have a particle with charge q_j. (Each charge can be either positive or negative.)

They want to study the total force on each particle, by measuring it and then comparing it to a computational prediction. This computational part is where they need your help. The total net force on particle j, by Couiomb's Law, is equal to

$$F_j = \sum_{i<j} \frac{Cq_iq_j}{(j-i)^2} - \sum_{i>j} \frac{Cq_iq_j}{(j-i)^2}.$$

Designing an $O(n \log n)$-time algorithm to compute F_j for all $j \in [n]$, using the Fast Fourier Transform algorithm. It suffices for you to reduce the problem to the convolution problem.