Homework 2

Course: Algorithm Design and Analysis **Semester**: Spring 2026

Instructor: Shi Li **Due Date**: **Apr 26, 2026**

Student Name: _____ Student ID: _____

Problems	1	2	3	4	5	6	7	8	Total
Max. Score	15	10	15	15	10	10	15	15	105
Your Score									

The sum of maximum scores over all problems is 105, but your score will be capped at 100.

For divide-and-conquer algorithms, you need to give the recurrence for the running time. For the greedy algorithm,

1 Divide and Conquer

Problem 1. Use the master theorem to provide asymptotic bounds for the following recurrences.

(1a) $T(n) = 4T(n/2) + O(n)$. $T(n) = O(\_\_\_\_\_)$

(1b) $T(n) = 4T(n/2) + O(n^2)$. $T(n) = O(\_\_\_\_\_)$

(1c) $T(n) = 3T(n/3) + O(n^2)$. $T(n) = O(\_\_\_\_\_)$

(1d) $T(n) = 7T(n/3) + O(n^2)$. $T(n) = O(\_\_\_\_\_)$

(1e) $T(n) = 8T(n/2) + O(n^3)$. $T(n) = O(\_\_\_\_\_)$

Problem 2. The master theorem you learnt does not cover the case when the non-recursive term contains logarithmic terms. Consider the recurrence $T(n) = 2T(n/2) + O(n \log^k n)$ for some integer constant $k \geq 1$. Prove that $T(n) = O(n \log^{k+1} n)$. You can use either the recursion tree method, or proof by induction.

Problem 3. In the standard inversion counting problem, we count pairs (i, j) such that $i < j$ and $A[i] > A[j]$. In the weighted inversion problem, every element i in the array has an associated weight $w_i \in \mathbb{Z}_{\geq 0}$. The weight of an inversion (i, j) is defined as $w_i w_j$. The goal of the problem is to compute the total weight of all inversions in the array.

Example Suppose the array is $A = [4, 1, 3]$ with weights $w = [10, 2, 5]$:

- The pair $(4, 1)$ is an inversion with weight $10 \times 2 = 20$.
- The pair $(4, 3)$ is an inversion with weight $10 \times 5 = 50$.
- So, the total weight of all inversions is $20 + 50 = 70$, which is the number you need to output.

Design a divide and conquer algorithm, that give the array $A[1..n]$ and weights $w_1, w_2, \cdots w_n \geq 0$, solves this problem in $O(n \log n)$ time.

Problem 4. Recall that in homework 1, you solved the largest rectangle in a histogram problem using the Union-Find data structure. Now, you need to design an $O(n \log n)$ algorithm to solve the same problem using a divide-and-conquer approach.

Example from Homework 1: Given heights $(3, 5, 10, 11, 20, 4, 8, 10)$, the largest rectangle has size 30, which is achieved by taking a height of 10 across a width of 3 (covering the columns with heights 10, 11, and 20).

2 Greedy Algorithms

Problem 5. For the Huffman code problem, we talked about the "even splitting" strategy and said that it is not safe. The strategy works as follows: split the letters into two groups such that the difference between the total frequencies of the two groups is minimized. Then we assign the first group of letters to the left subtree (i.e., the first bit of the encodings for these letters is 0), and the second group of letters to the right subtree (i.e., the first bit of the encodings for these letters is 1). Show that the strategy is not safe by providing a counter-example where the even-splitting decision will miss all optimum solutions.
Hint: Try to start from the smallest example. Is there a counter-example with 3 letters? Is there a counter-example with 4 letters?

Problem 6. You are given n files of various sizes $\{s_1, s_2, \ldots, s_n\}$. You want to merge them into a single file by merging two files at a time. The cost of merging two files of size a and b is $a + b$.

Example: Suppose you have three files with sizes $\{10, 20, 30\}$.

- If you first merge 10 and 20, the resulting file has size 30 and the cost is $10 + 20 = 30$. Then, merging this new file with the remaining 30 size file costs $30 + 30 = 60$. Total cost: $30 + 60 = 90$.
- If you first merge 20 and 30, the cost is 50. Merging the result with 10 costs $50+10 = 60$. Total cost: $50 + 60 = 110$.

In this case, the first merging pattern is more efficient. Design a greedy algorithm to solve the problem.

Problem 7. Consider a set of n boxes, each with a capacity of 1, and m items where each item i has a size $s_i < 1$. We are restricted to putting **at most 2 items** into any single box. A set of 1 or 2 items can be put into a box if their total size is at most 1. The goal is to maximize the total number of items placed into the n available boxes.

Example: If we have 2 boxes and items with sizes $\{0.9, 0.8, 0.2, 0.2\}$, we can pack 3 items by putting items with sizes 0.8 and 0.2 on one box, and the other item of size 0.2 into the other box.

Design a greedy algorithm to solve this problem.

Problem 8. Consider a project that needs to be run on a supercomputer and infinite number of PCs. The project contains n distinct jobs, indexed as $[n]$. Each job $j \in [n]$ consists of two stages: first it needs to be preprocessed on the supercomputer, and then it needs to be finished on one PC. Say a job $j \in [n]$ needs p_j seconds of time on the supercomputer, followed by f_j seconds of time on a PC.

Since there are infinite number of PCs, the finishing of the jobs can be performed fully in parallel. However, there is only one supercomputer, and it can only work on a single job at a time. You need to find an order to feed the jobs to the supercomputer. As soon as the first job in order is done on the supercomputer, it can be handed off to a PC for finishing; at that point in time the second job in order can be fed to the supercomputer, and so on. The completion time of order is the earliest time for which all jobs will have finished processing on the PCs.

Design a polynomial-time algorithm that finds a schedule with the smallest completion time.

For example, for the instance given in Table 1, the optimum order have a completion time of 160, given in Table 2.

j	1	2	3	4
p_j	40	50	20	30
f_j	50	20	70	60

Table 1: The scheduling instance with 4 jobs.

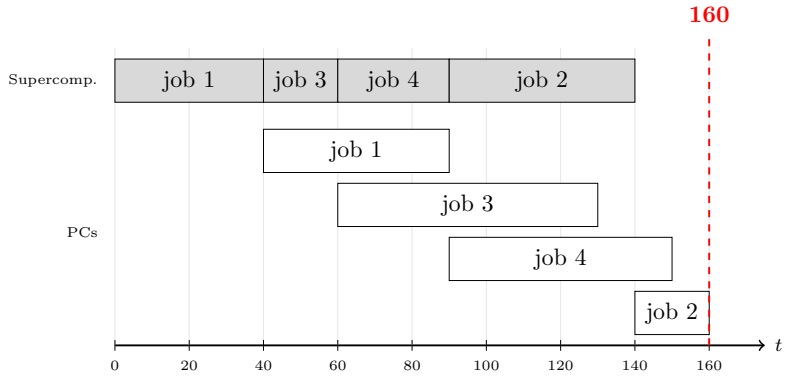

Table 2: The schedule using the order $1, 3, 4, 2$.