

Name: \_\_\_\_\_ Wisc ID: \_\_\_\_\_

**Ground Rules**

- Answer the questions in the boxes provided on the question sheets. If you run out of room for an answer, add a page to the end of the document. **Since students have had issues fitting their answers into the boxes, we are increasing the size of the boxes for this problem set. However, do NOT feel obligated to fill the entire solution box. The size of the box does NOT correspond to the intended solution length.**
- The homework is to be done and submitted individually. You may discuss the homework with others in either section but you must write up the solution *on your own*.
- You are not allowed to consult any material outside of assigned textbooks and material the instructors post on the course websites. In particular, consulting the internet will be considered plagiarism and penalized appropriately.
- The homework is due at 11:59 PM CST on the due date. No extensions to the due date will be given under any circumstances.
- Homework must be submitted electronically on Gradescope.

### Problem 1:

1. In the year 2048 (when teleportation is already invented), you have opened the first delivery company (*FastAlgo Express*) that uses a teleportation machine to deliver the packages. Suppose on a certain day,  $n$  customers give you packages to deliver. Each delivery  $i$  should be made within  $t_i$  days and the customer pays you a fixed amount of  $p_i$  dollars for doing it on time (if you don't do it on time, you get paid 0 dollars). On-time delivery means that if package  $i$  is due within  $t_i = k$  days, we should deliver it on one of the days  $1, 2, \dots, k$ , to be on time. Unfortunately, your teleportation machine is able to do at most **one** delivery to **one** destination in one day (you cannot deliver packages to multiple destinations within 1 day). Your goal is to figure out which deliveries to make and when.

**Input:** A set of  $n$  deliveries with due dates  $t_i \in \mathbb{N}, t_i \geq 1$  and payments  $p_i > 0$  for each delivery  $i \in \{1, \dots, n\}$ .

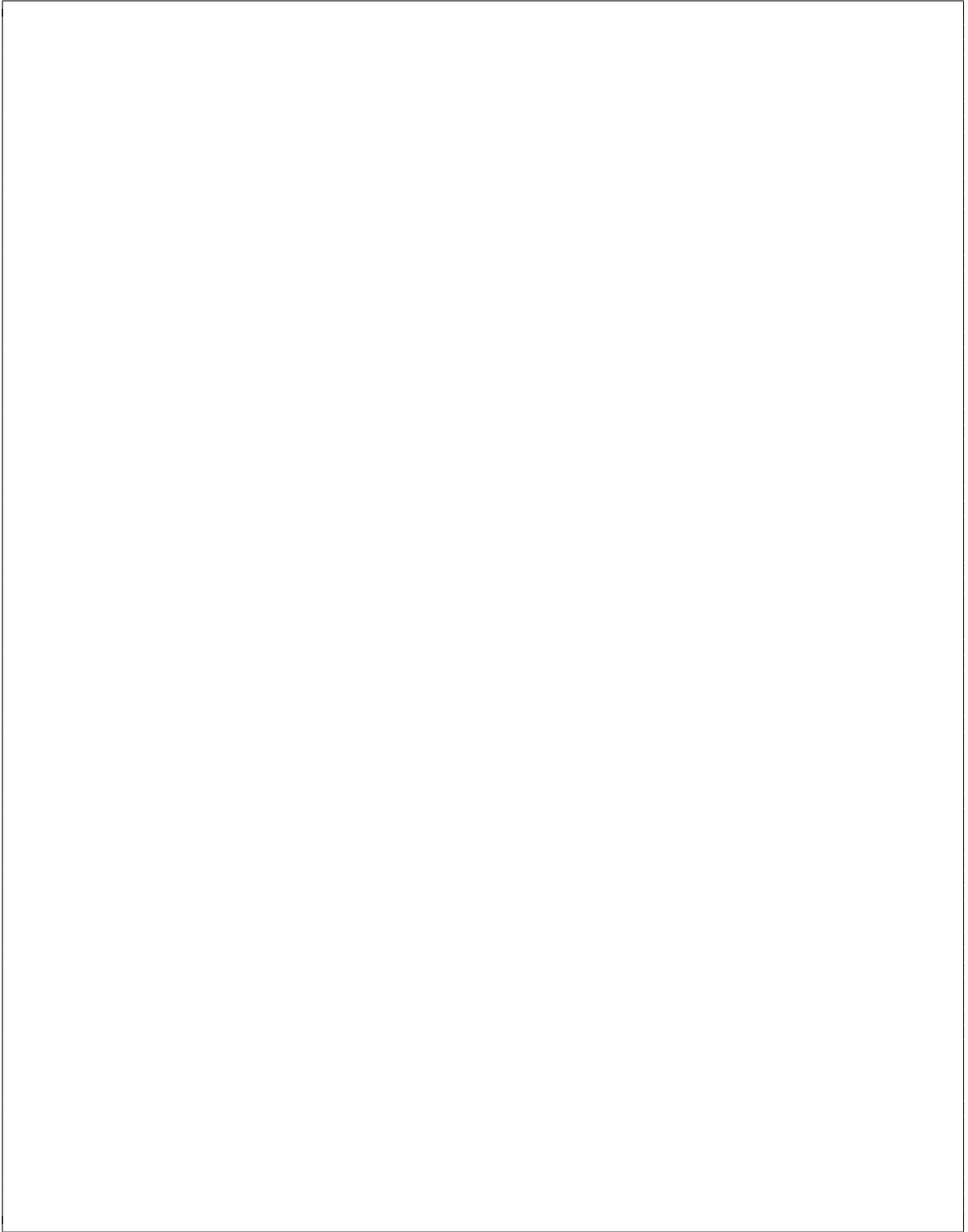
**Output:** A delivery order such that your company's profit (the sum of  $p_i$  for the deliveries made on time) is maximized.

**Example:** You have 4 deliveries with due dates:  $t_1 = 2, t_2 = 3, t_3 = 1, t_4 = 2$ . You cannot deliver all of them on time, but you can make the third delivery on day 1, then the first on day 2 and the second on day 3.

We call a subset  $S$  of the deliveries *feasible* if there is a way to order the deliveries in  $S$  so that each one is made on time. You may assume the following lemma holds without proving it:  *$S$  is feasible if and only if for all days  $t$  that  $t \leq n$ , the number of deliveries in  $S$  due within  $t$  days is no more than  $t$ .*

Describe and analyze an efficient algorithm to determine the order in which the deliveries should be made so that you maximize your profit. You should aim for a runtime  $O(n^2)$  or better.

(Hint: Build up a feasible set of deliveries by considering them in a particular order and adding each delivery to your schedule if the set of selected deliveries continues to remain feasible. Use the lemma to check for feasibility. What order should you consider deliveries in to maximize the profit? Use an "exchange" argument to prove the correctness of your algorithm.)



## Problem 2:

2. There are  $m$  doors numbered 1 to  $m$  which are all initially closed. Your goal is to open them by pressing  $n$  switches numbered from 1 to  $n$  in an order of your choice. The behavior of the switches is described by an  $n \times m$  matrix  $M$ . When switch  $i$  is pressed door  $j$  behaves as follows:

- If the  $(i, j)$ th entry,  $M_{ij}$ , is 1, door  $j$  **opens** no matter whether it was open or closed before.
- If the  $(i, j)$ th entry,  $M_{ij}$ , is  $-1$ , door  $j$  **closes** no matter whether it was open or closed before.
- If the  $(i, j)$ th entry,  $M_{ij}$ , is 0, door  $j$  **remains at its previous state** (open or closed).

In this problem, you will design a greedy algorithm for determining an order to press the switches assuming such an order exists. **Each switch must be pressed exactly once.** Your algorithm is given the matrix  $M$  as input.






**Example** Suppose there are  $m = 5$  doors and  $n = 3$  switches with matrix  $M$  shown on the right. The optimal way to open all doors is to press switches in the order 2,3,1.

- Initially: (Closed, Closed, Closed, Closed, Closed)

- After 2: (Open, Open, Open, Closed, Closed)

- After 3: (Open, Closed, Open, Closed, Open)

- Finally: (Open, Open, Open, Open, Open)

					
Switch <sub>1</sub>	0	1	1	1	0
Switch <sub>2</sub>	1	1	1	0	-1
Switch <sub>3</sub>	0	-1	0	0	1

- (a) Determine an ordering that works for the following matrix. No explanation is required.

**Hint:** Think about which switch should be pressed last.

$$\begin{bmatrix} 1 & -1 & 1 & -1 \\ 0 & 1 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & -1 & 1 \end{bmatrix}$$

- (b) Now consider a general instance with  $n$  switches and  $m$  doors. Assume that all the doors can open by pressing the switches in some order. State a greedy rule for determining which of the  $n$  switches should be pressed **last**.

- (c) Briefly prove the correctness of your greedy rule. That is, show that if there exists an ordering of switches for opening all doors, then there exists one where the switch you chose in **part (b)** is pressed last.

- (d) Develop an algorithm for determining an ordering (in reverse order from last to first) based on your greedy rule from **part (b)**. Describe your algorithm by filling out details in the following pseudocode. Here  $S$  denotes the set of switches left to consider when some partial suffix of the ordering has been determined. Feel free to use any other notation your algorithm requires.

---

**Algorithm 1:** Order( $M$ )

Returns an ordering over  $1, \dots, n$  that opens all doors, in reverse order, given the matrix  $M$ .

---

1 Initialize  $S = \{1, \dots, n\}$ .

2 /\*Add any extra bookkeeping\*/

3 **while**  $S \neq \emptyset$  **do**

4     /\*Insert greedy rule here to determine an index  $i$ .\*/

5     Append  $i$  to the end of the ordering.

6     Set  $S := S \setminus \{i\}$ .

7     /\*Add any extra bookkeeping\*/

8 **Return** ordering.

---

- (e) State the asymptotic runtime of your algorithm in **part (d)**. No explanation required.