CS570 Spring 2025: Analysis of Algorithms Exam II

	Points		Points
Problem 1	16	Problem 4	17
Problem 2	9	Problem 5	18
Problem 3	20	Problem 6	18
			Total 98

First name	
Last Name	
Student ID	

Instructions:

1. This is a 2-hr exam. Closed book and notes. No electronic devices or internet access.
2. A single double sided 8.5in x 11in cheat sheet is allowed.
3. If a description to an algorithm or a proof is required, please limit your description or proof to within 150 words, preferably not exceeding the space allotted for that question.
4. No space other than the pages in the exam booklet will be scanned for grading.
5. If you require an additional page for a question, you can use the extra page provided within this booklet. However please indicate clearly that you are continuing the solution on the additional page.
6. Do not detach any sheets from the booklet.
7. If using a pencil to write the answers, make sure you apply enough pressure, so your answers are readable in the scanned copy of your exam.
8. Do not write your answers in cursive scripts.
9. This exam is printed double sided. Check and use the back of each page.

Student ID:

1) 16 pts (2 pts each)

Mark the following statements as **TRUE** or **FALSE** by circling the correct answer. No need to provide any justification.

[TRUE/FALSE]

Bellman-Ford is asymptotically faster than Dijkstra's algorithm when dealing with graphs that only have non-negative edge weights.

[TRUE/FALSE]

In a maximum flow problem in a flow network with non-zero integer capacities, there must exist at least one integer-valued maximum flow f with non-zero $v(f)$.

[TRUE/FALSE]

Given a feasible circulation in a network with unlimited capacities, if the flow on each edge is increased by one unit, we will get another feasible circulation.

[TRUE/FALSE]

Suppose an edge e is not saturated due to max flow f in a Flow Network. Then increasing e's capacity will not increase the max flow value of the network.

[TRUE/FALSE]

The worst-case time complexity of any dynamic programming algorithm with n^3 unique subproblems is $\Omega(n^3)$.

[TRUE/FALSE]

Consider the following recurrence formula

$$OPT(j) = \max_{1 \le i \le j} \{A[i] - B[j - i] + OPT(j - i)\}$$

where A and B are fixed input arrays and subproblems OPT(j) are defined for $j = 1, \ldots, n$. Then, this will lead to an O(n) dynamic programming algorithm.

[TRUE/FALSE]

By definition, all dynamic programming algorithms must run in polynomial time with respect to the input size (either in terms of the number of integers, or in terms of the number of bits in the input).

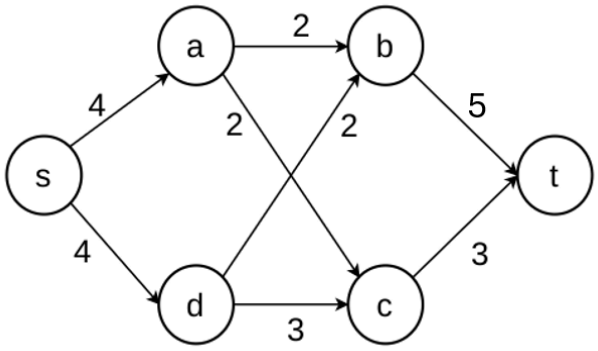
[TRUE/FALSE]

Given a flow network G and its max flow f, we can always find an s-t cut (A,B) in G where all edges e crossing the cut either have $f(e) = C(e)$ or $f(e) = 0$.

Student ID:

2) 9 pts - 3 pts each. No partial credit

I. What is the capacity of the minimum s-t cut in the following flow network?



A) 5
B) 6
C) 7
D) 8
Solution: C

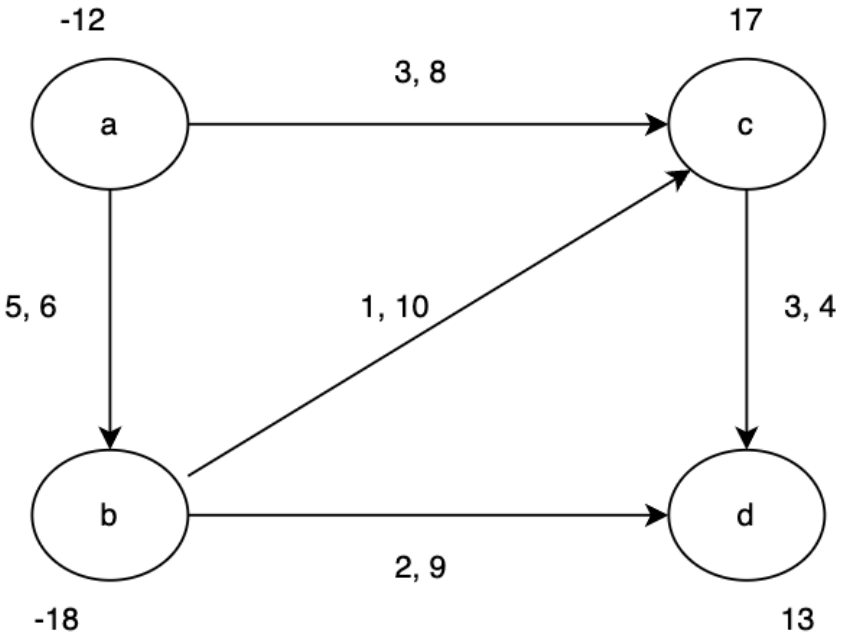
II. Which of the following statements is/are <u>False</u> regarding dynamic programming? Select all correct answers!
A) It solves problems by breaking them into overlapping subproblems.
B) It can be implemented using a bottom-up approach known as tabulation.
C) A problem of size n must have subproblems of size n/b where b>1 is an integer constant
D) It can utilize memoization to store values of optimal solutions for unique subproblems to avoid redundant calculations.
Solution: C

III. Which of the following statements is/are <u>True</u> about the Ford-Fulkerson algorithm? Select all correct answers!
A) For integer edge capacities, it terminates in at most $v(f)$ iterations, where $v(f)$ is the maximum flow value.
B) For integer edge capacities, it terminates in at most C iterations, where C is the total capacity of edges out of source.
C) It may converge to an incorrect value of max flow for non-integer edge capacities.
D) Assuming it terminates, it will always find the same maximum flow f independent of the choice of augmenting paths.
Solution: A,B,C

Student ID:

3) 20 pts

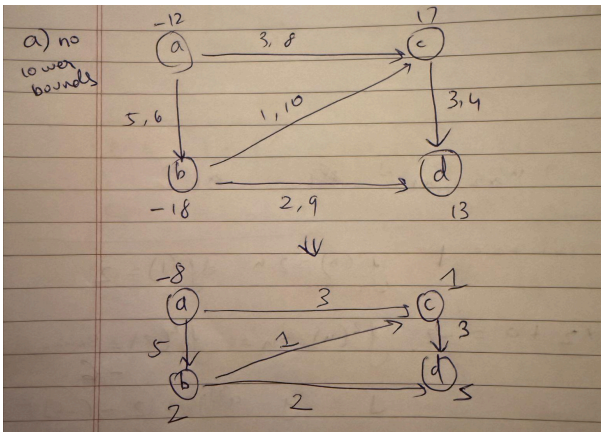
Consider the flow network below with demands and lower bounds.
Notation: (l_e, C_e) show lower bound and capacity on each edge. Numbers assigned to nodes show the demand value at that node.



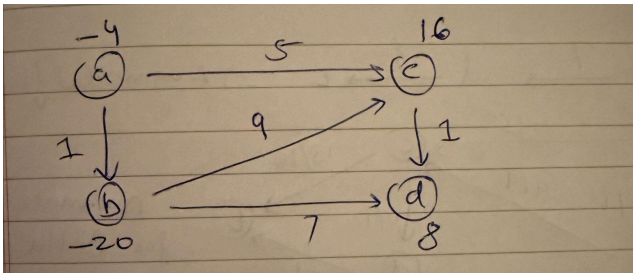
Follow the steps as described in class to determine if a feasible circulation exists

a) Convert the given network to an equivalent one with no lower bounds. Show all your work. (6 pts)

No lower bounds



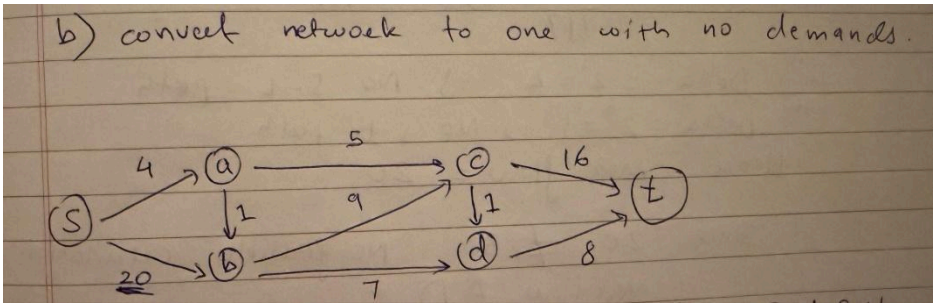
Student ID:



RUBRIC:
(3 points) Draw the imbalance graph correctly
(3 points) Draw the final graph with no lower bounds correctly
-1 point for every wrong number

b) Convert the network obtained in a) to an equivalent one with no demands (i.e., a flow network) (2 pts)

Flow Network (i.e., with no demands)

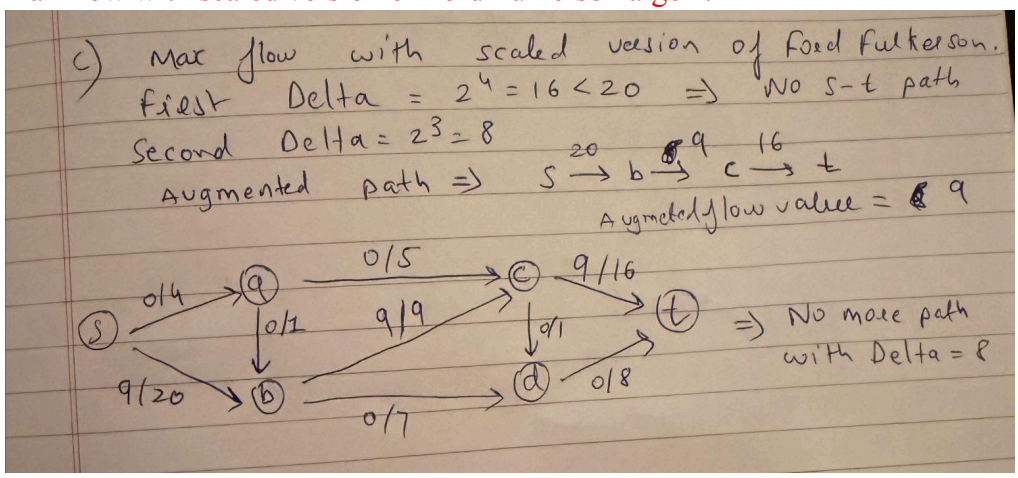


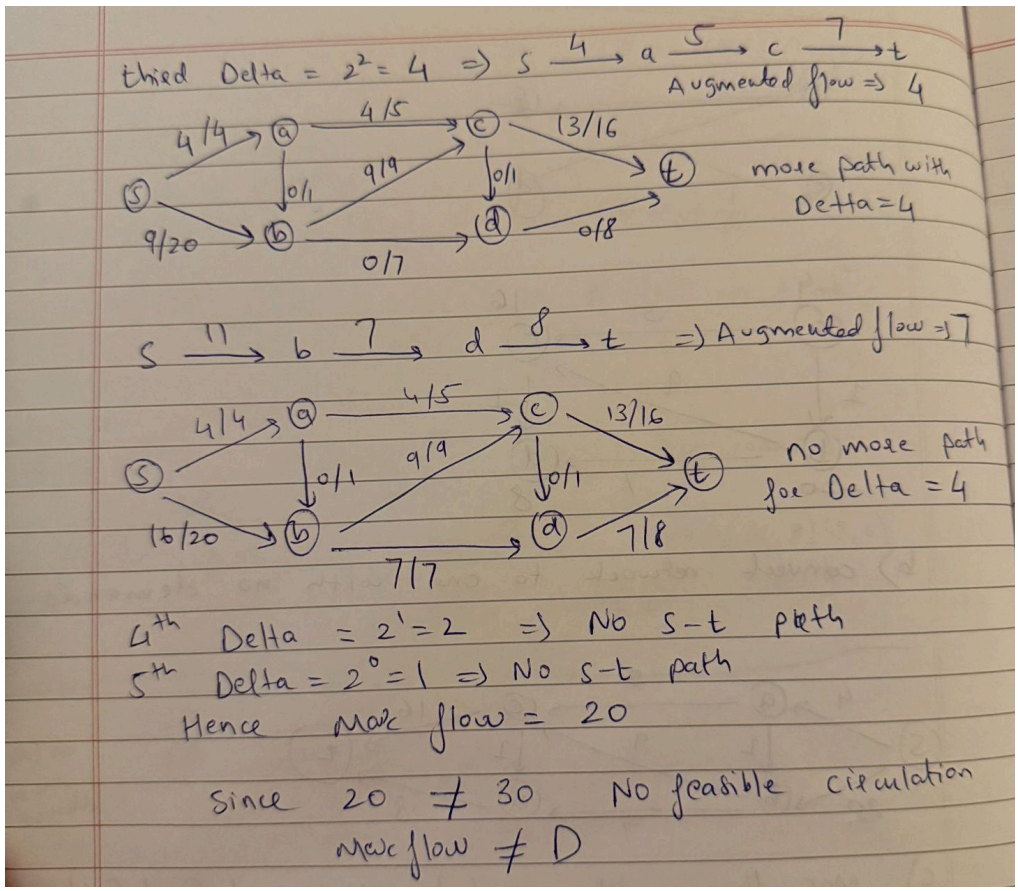
RUBRIC:
(1 point) Create a super source and super sink
(1 point) Connect nodes correctly to either source/sink
0 points if no super source or super sink was created

c) In the network obtained in b), compute max flow using Scaled version of Ford Fulkerson algorithm. Only show the different scaling iterations marked with the corresponding Delta value, and for each scaling iteration, show the augmenting path(s) found with the respective augmenting flow value. No need to show residual graphs. (10 pts)

Student ID:

Max flow with scaled version of Ford Fulkerson algorithm





RUBRIC:

(- 5 points): Did not use scaled version of Ford Fulkerson algorithm
(- 2 points): Incorrect max flow value
(- 2 points): Missed delta 8, 4 and 1 in calculation
(- 1 point): For each minor calculation error

Student ID:

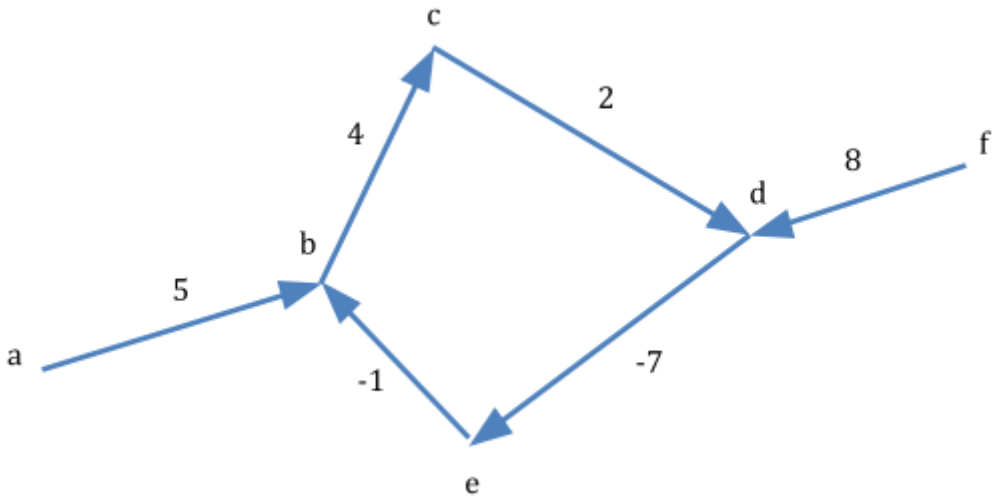
d) Based on your solution in part c, explain whether the originally given network has a feasible circulation or not. (2 pts)
Since max flow not equal to total demand, the circulation is not feasible

(2 points) correctly mention that circulation is not feasible because max flow not equal to total demand
0 points if there is no mention of max flow being not equal to demand **AND** circulation not being feasible

Student ID:

4) 17 pts

Consider the directed graph below.



a) Use the Bellman-Ford algorithm to find the negative cycle in this graph. In other words, you must solve this problem numerically with the given edge weights and Explain how the destination node t is determined (3 pts)

create new node t* and connect all nodes to t* with distance of 0.

1 - point for creating new node
2 - for connecting all the nodes distance 0 (or any number as long as it is the same for all nodes)
0 - no creation of a new node

b) State how many iterations of Bellman-Ford are required (2 pts)

n=7 iterations
2 - Point for stating the correct number of iterations
0 - Incorrect answer

Student ID:

c) Show each iteration of Bellman-Ford by showing subproblem values in a table with columns representing nodes and rows representing iterations (6 pts)

a	∞	0	0	0	0	0	0	0
b	∞	0	0	0	-1	-2	-2	-2
c	∞	0	0	-5	-6	-6	-6	-7
d	∞	0	-7	-8	-8	-8	-9	-10
e	∞	0	-1	-1	-1	-2	-3	-3
f	∞	0	0	0	0	0	0	-1
t*	0	0	0	0	0	0	0	0
	w/ up to 0 edges	w/ up to 1 edges	w/ up to 2 edges	w/ up to 3 edges	w/ up to 4 edges	w/ up to 5 edges	w/ up to 6 edges	w/ up to 7 edges

1 - Point for each correct iteration. **Whole** iteration must be correct to get 1 full point.

d) Show how it is determined that a negative cycle exists using your results (2 pts)
Some of the numbers in **iteration 7** still go down. This is an indication that a shorter path using (n=7 edges) to the destination exists in the graph. Since the graph only has n=7 nodes, this path must contain a negative cycle.

2 - Points for correctly stating the iteration number: 7 and the logic behind selecting iter 7.

e) Show how the negative cycle is found using your calculated results (4 pts)
We start from any of the nodes where the distance goes down at iteration 7 and find the shortest path (containing negative cycle) to t*.
e.g. start at d.
- d's shortest distance to t* comes from e
- e's shortest distance to t* comes from b
- b's shortest distance to t* comes from c
- c's shortest distance to t* comes from d

We have come back to the starting point of the cycle. Therefore debcd forms a negative cycle.

1 - Point for stating that distance goes down at **7th Iteration**
2 - Points for explaining the shortest distance method and the loop.
1- Point for giving the correct cycle : debcd
0 - If does not start from iteration 7.

Student ID:

5) 18 pts

Suppose USC wants to send m teams of students into a coding competition, each team consisting of one freshman and one sophomore (a student cannot be on multiple teams). Selected students are a set of freshmen $\{s_1 \ldots s_m\}$ and sophomores $\{s_{m+1} \ldots s_{2m}\}$ who need to be matched to form teams. Further, each team formed needs to be assigned a professor to mentor them. For this, a set of professors $\{p_1 \ldots p_n\}$ are available, but to ensure some load-balancing, each professor can mentor up to 5 teams each and must mentor at least one team. Each student s_i (i = 1, …, 2m) is asked for a subset of professors X_i they'd like to be mentored by, and a professor p_k can be assigned to mentor a team (s_i, s_j) if p_k is in both X_i and X_j.

a) Design a network flow algorithm to determine if it is feasible to form *m* teams and assign mentors given all the constraints above. (12 points)

We will have 3 *layers* - freshmen, sophomores, and professors. The idea would be to have a 'unit of flow' represent a team with its mentor. Professor layer has to be in the middle so that it can connect to both the other layers, which is required to capture the connectivity constraints posed by the X_i's.

Network construction:
1) Node S with demand -m, node T with +m.
2) First layer $\{s_1 \ldots s_m\}$ as nodes, with edges from S to all with cap 1 (optional: lb 1),
3) Second layer p_j's as **EDGES** with lb 1, cap 5,
4) 3rd layer $\{s_{m+1} \ldots s_{2m}\}$ as nodes, with edges to T from all with cap 1 (optional: lb 1).
5) Add edge between a professor p_k and a student s_i (directed along S to T) if p_k is in X_i.

Alternative with no S/T:
No need to add S,T (and their edges) - Directly add demand -1 to freshmen nodes, and +1 to sophomore nodes.

the freshmen and sophomore nodes have symmetric design and their constructions can be flipped (i.e. all arrow directions and demands)

Rubric:
2 pts - representing freshmen and sophomore as nodes
3 pts - Nodes S, T (with edges to corresponding student layers) and demands -m, +m
OR simply adding demands -1, +1 to all the freshmen, sophomores resp.
4 pts - Points for professors represented as edges (2) with correct LB and cap (2)
3 pts - Points for connecting EACH of the student layers to the professor layer (2) and mentioning s_i is connected to p_k if it is contained in X_i (1)..

Student ID:

b) Prove correctness of your algorithm. (6 points)

Forming *m* teams and assigning mentors given all the constraints is feasible if and only if the constructed network has feasible circulation.

=>) Suppose m teams with mentors can be formed to satisfy all the constraints.
For each team (s_i, s_j) and their mentor p_k , assign flow of 1 on the path S - s_i - edge p_k - s_j - T (if using the 'no S/T approach', simply the path s_i - edge p_k - s_j).
This path exists as per the step 5) of construction since p_k must be in both Xi and Xj.
Each student being in exactly 1 team (thus, having m total teams) satisfies the demands on S and T and the capacities of the corresponding edges (or demands on all the student nodes if using 'no S/T approach').
Load-balancing of mentors ensures that LB and cap on each edge p_k is satisfied.
Thus, we have feasible circulation in the network.

<=) Suppose the network has feasible circulation.
For every unit flow on an edge p_k , we can trace it to either side using flow conservation (since no demands on endpoints of edge p_k and the student nodes) to get a path S - s_i - edge p_k - s_j - T with unit flow (if using the 'no S/T approach', simply the path s_i - edge p_k - s_j using flow conservation on endpoints of edge p_k). For each such path, form the team (s_i, s_j) and assign p_k as their mentor.
step 5) of construction ensures p_k must be in both X_i and X_j.
Demands on S, T ensure there are m teams. Capacities 1 on edges from S to freshmen and sophomores to T ensure each student is in exactly one team. (ensured by demands on all the student nodes if using 'no S/T approach').
Load-balancing of mentors is ensured by the LB and cap on each edge p_k.

2 - Point for claim that's clear and precise (for a nearly correct network)
2 - Points for proof in forward direction
2 - Points for proof in backward direction

Student ID:

6) 18 pts

A film screening festival is being hosted in your neighborhood which will last for D days, with one movie screened each day. The list of movies consists of English, French and Korean movies.

The movie screened on day d

- is in language $L_d \in$ {En, Fr, Ko},
- has a runtime of t_d minutes (integer) and
- has a rating of R_d (may not be integer).

Given your busy schedule, you have decided you can spend at most T time for the film festival (in minutes, integer). You want to watch the best movies as much as possible, so you set the objective of maximizing the total rating of movies you watch. Your only other constraint is that you do NOT want to watch two movies in the same language back to back. For instance, if you watch a movie on day 10 and day 14 (skipping days 11-13), then L_{10} and L_{14} must be different languages.

Use dynamic programming to compute the maximum total rating of movies you can watch given the constraints above.

Define (in plain English) subproblems to be solved. (4 pts)

OPT_en(d,t) = Max total rating possible from watching movies from days 1, …, d with t minutes available, with the last movie watched being in English. Similarly, Opt_fr(d,t) and Opt_ko(d,t).

Equivalently, *OPT(d, t, l)* = Max total rating possible from watching movies from days *1, …, d* with *t* minutes available, with the last movie watched being in Language *l*.

4pt - correct subproblem

2pt - (if not correct) close enough subproblem, including the following cases:

1. Miss to separate different languages for 2-D definition.
2. Define $OPT[d,\ t, L_d]$ with the last parameter specified to the exact language on day d.

Note that the two solutions above only differ in notation, not the actual subproblems.

b) Write the recurrence relation(s) for the subproblems (6 pts)

There should be a way to decide if you can watch the movie on day d by controlling which language is permissible.

If $L_d == En$**:**

$OPT_{en}[d,\ t] = max\{OPT_{en}[d-1,\ t],\ R_d + max\{OPT_{fr}[d-1, t-t_d],\ OPT_{ko}[d-1, t-t_d]\}\}$

If $L_d != En$**:**

$OPT_{en}[d,t] = OPT_{en}[d-1,t]$

Similar recurrence for $OPT_{fr}[d,t]$ and $OPT_{ko}[d,t]$ **(Condition checks for respective language *L* to include the 2nd term inside 'max' which calls subproblems for *L'* and *L"*, ignores the 2nd term if condition not true)**

Rubrics breakdown:

3pt - Correct for watching film on day d.

Only when $L_d != L_{d-1}$,

-- 2-Dimension

$OPT_{en}[d,t] = R_d + max\{OPT_{fr}[d-1, t-t_d],\ OPT_{ko}[d-1, t-t_d]\}$ similar for $OPT_{fr}[d,t], OPT_{ko}[d,t]$

-- 3-Dimension

$OPT[d,t,l] = R_d + \max_{ll \in \{En, Fr, Ko\}\ \&\&\ ll != l} \{OPT[d-1, t-t_d, ll]\}$

3pt - Correct for skipping case.

-- 2-Dimension

$OPT[d,t] = OPT[d-1,t]$ for each language

-- 3-Dimension

$OPT[d,t,l] = OPT[d-1,t,l]$

0pt - if movies are skipped without checking the conditions.

c) Using the recurrence formula in part b, write pseudocode using iteration to compute the minimum number of packages to meet the objective. (6 pts)

2pt - Correct pseudocode (order of computation (loop) increasing in d, t)
Precondition: OPT definition in a) and recurrence formula in b) have to be close to correct! - 0pt otherwise
1pt - each minor error (if more than one, 0pt)

Student ID:

Specify base cases and their values (3 pts)
OPT_en[0,t] = OPT_fr[0,t] = OPT_ko[0,t] = 0
OPT_en[i,t] = -inf for t<0 // okay to skip this base case if otherwise accounted for by a condition in the recurrence, that ignores a term when t_d > t etc.

1pt - Correct base case with zero available day. (no partial credit)
-- 2-Dimension
$\forall t \in [0,T],\ OPT_{en}[0,t] = OPT_{fr}[0,t] = OPT_{ko}[0,t] = 0$
-- 3-Dimension
$\forall t \in [0,T], l \in \{En, Fr, Ko\},\ OPT[0,t,l] = 0$

2pt - Correct base case with negative available time.
-- 2-Dimension
$\forall t < 0, OPT_{en}[d,t] = OPT_{fr}[d,t] = OPT_{ko}[d,t] = -\infty$
-- 3-Dimension
$\forall t < 0,\ OPT[d,t,l] = -\infty$

Specify where the final answer can be found? (1 pt)

1pt - Final answer location. (no partial credit)
-- 2-Dimension
$max\{OPT_{en}[D,T],\ OPT_{fr}[D,T],\ OPT_{ko}[D,T]\}$
-- 3-Dimension
$\max_{l \in \{En,Fr,Ko\}} \{OPT[D,T,l]\}$

d) What is the complexity of your solution? (1 pt)
Is this an efficient solution? (1 pt)
O(DT).
Pseudo-polynomial run time (due to non-polynomial dependence on the value T), i.e., this is not an efficient solution.

1pt - Correct Time Complexity O(D*T).
1pt - Stating not efficient