THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 320: MIDTERM EXAMINATION – November 8, 2021

Full Name: ______________________ CS Ugrad ID: _______________

Signature: ______________________ UBC Student #: |_|_|_|_|_|_|_|_|

Important notes about this examination

1. You have 90 minutes to complete this exam.
2. ONE double-sided, letter-size note sheets are allowed.
3. All original pages of this booklet must be returned when you finish your exam.
4. Good luck!

Student Conduct during Examinations

1. Each examination candidate must be prepared to produce, upon the request of the invigilator or examiner, his or her UBCcard for identification.
2. Examination candidates are not permitted to ask questions of the examiners or invigilators, except in cases of supposed errors or ambiguities in examination questions, illegible or missing material, or the like.
3. No examination candidate shall be permitted to enter the examination room after the expiration of one-half hour from the scheduled starting time, or to leave during the first half hour of the examination. Should the examination run forty-five (45) minutes or less, no examination candidate shall be permitted to enter the examination room once the examination has begun.
4. Examination candidates must conduct themselves honestly and in accordance with established rules for a given examination, which will be articulated by the examiner or invigilator prior to the examination commencing. Should dishonest behaviour be observed by the examiner(s) or invigilator(s), pleas of accident or forgetfulness shall not be received.
5. Examination candidates suspected of any of the following, or any other similar practices, may be immediately dismissed from the examination by the examiner/invigilator, and may be subject to disciplinary action:
   i. speaking or communicating with other examination candidates, unless otherwise authorized;
   ii. purposely exposing written papers to the view of other examination candidates or imaging devices;
   iii. purposely viewing the written papers of other examination candidates;
   iv. using or having visible at the place of writing any books, papers or other memory aid devices other than those authorized by the examiner(s); and,
   v. using or operating electronic devices including but not limited to telephones, calculators, computers, or similar devices other than those authorized by the examiner(s)—(electronic devices other than those authorized by the examiner(s) must be completely powered down if present at the place of writing).
6. Examination candidates must not destroy or damage any examination material, must hand in all examination papers, and must not take any examination material from the examination room without permission of the examiner or invigilator.
7. Notwithstanding the above, for any mode of examination that does not fall into the traditional, paper-based method, examination candidates shall adhere to any special rules for conduct as established and articulated by the examiner.
8. Examination candidates must follow any additional examination rules or directions communicated by the examiner(s) or invigilator(s).

1 236413 048929

This page intentionally left (almost) blank.
Do not write answers here; there is an extra page at the end.

CPSC 320 2021W1: Midterm Exam 2

November 8, 2021

1 ID Please (1 mark)

Write your CS undergraduate login ID again in this box:

2 Mystery Recurrence (11 marks)

An algorithm that solves a mystery problem has runtime described by the following recurrence, where $c > 0$:

$$T(n) = \begin{cases} c, & \text{for } n \leq 3, \\ 2T(n/3) + T(n/4) + cn^2, & \text{for } n > 3. \end{cases}$$

1. (3 marks) Draw or describe the first two levels (i.e., the root at level 0 and the nodes at level 1) of the recursion tree for this recurrence. Label each node with the problem size and the time to solve the mystery problem, not counting recursive calls.

2. (2 marks) What is the total time (not counting recursive calls) for nodes at level 1? No justification needed.

3. (2 marks) What is the total time (not counting recursive calls) for nodes at level i, assuming that there is no leaf at a level $\leq i$? No justification needed.

4. (3 marks) Which of these expressions is an *upper bound* on the runtime of the algorithm? **Check all that apply.**

   ◯ $O(n^2)$ ◯ $O(n^2 \log n)$

   ◯ $O(n^3)$ ◯ $O(3^n)$

5. (3 marks) Which of these expressions is a *lower bound* on the runtime of the algorithm? **Check all that apply.**

   ◯ $\Omega(n^2)$ ◯ $\Omega(n^2 \log n)$

   ◯ $\Omega(n^3)$ ◯ $\Omega(3^n)$

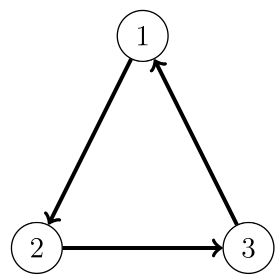
3 Finding a Place in the Pecking Order (16 marks)

This question continues the Pecking Order problem from the first midterm. To refresh your memory, here is the problem description from Midterm 1:

Small clusters of domesticated chickens typically maintain a pairwise pecking order: for *every* pair of chickens i and j, one dominates the other when it comes to pecking rights at food time.

A dominance relation can be represented by a *dominance graph* in which nodes represent chickens and a directed edge from i to j indicates that i dominates j. In a dominance graph, for every pair of distinct nodes i and j, there is a directed edge from i to j, or from j to i, but not both. A *pecking order* of $n \geq 1$ chickens is a permutation $L(1), \ldots, L(n)$ of the chickens such that for $1 \leq i \leq n-1$, chicken $L(i)$ dominates chicken $L(i+1)$.

For example, with three chickens where 1 dominates 2, 2 dominates 3, and 3 dominates 1, one pecking order is 1,2,3 and the dominance graph is



Midterm 1 gave an algorithm to compute a full pecking order of a dominance graph G with $n \geq 1$ nodes.

```
function PECKING-ORDER(G)                    ▷ G is a dominance graph with n ≥ 1 nodes (chickens)
    L ← (1)                                  ▷ L is initialized to be the list containing chicken 1
    i = 2
    good ← true
    while (i ≤ n) and good do                ▷ try to insert chicken i into the current pecking order L
        if L(i − 1) dominates i then         ▷ L(i − 1) is currently the last item in list L
            insert i at the end of the list L
        else if i dominates L(1) then        ▷ L(1) is the first item in list L
            insert i at the start of the list L
        else                                 ▷ L(1) dominates i and i dominates L(i − 1)
            good ← false
            k ← 1
            while (not good) and (k < i − 1) do
                if L(k) dominates i and i dominates L(k + 1) then
                    insert i between L(k) and L(k + 1)
                    good ← true
                end if
                k ← k + 1
            end while
        end if
        i ← i + 1
    end while
    if good then
        return the list L
    else
        return "No full pecking order found"
```

27: **end if**
28: **end function**

On **this** midterm, we will modify this algorithm slightly. In particular, we replace lines 11–19 of the above code with the following:

11: $k \leftarrow$ FindPeckingPlace$(i, 1, i-1)$
12: insert i between $L(k)$ and $L(k+1)$

where FindPeckingPlace is defined (except for the missing part that you need to figure out) as:

1: **function** FindPeckingPlace(i,low,high)
// This function should compute an index k with low $\leq k <$ high such that
// $L(k)$ dominates i, and i dominates $L(k+1)$.
// It should only be called with low $<$ high, and when
// L(low) dominates i, and i dominates L(high).
2: **if** low $+ 1 =$ high **then**
3: **return** low ▷ Base Case: $k =$ low is a suitable index.
4: **else**
*** MISSING CODE ***
5: **end if**
6: **end function**

Your job is to supply the "MISSING CODE" to make the function work correctly. For full credit, your solution must (1) work correctly, (2) run in $O(\log n)$ time, where $n =$ high $-$ low, and (3) be recursive. You are NOT allowed to modify anything else in the code. Write your answer below. You do not need to justify your code, but doing so may help you earn credit even if your code isn't sufficiently clear or correct.

4 Study Plan: Part I (19 marks)

You have exams in C different courses, and have n total hours to prepare. Your grade on any exam depends on how many hours you spend studying for said exam. You have a 2D array $M[1..C][0..n]$, where for $1 \le c \le C$, $M[c][0]$ is the mark you'll get in course c if you don't study for it at all, and for $1 \le j \le n$, $M[c][j]$ records the additional points you'll get from the jth hour of study for course c. *Assume throughout this problem that for any given course, the additional marks gained from the jth hour of study is at least the marks gained from the $(j+1)$st hour of study.* That is,

$$M[c][j] \ge M[c][j+1], \text{ for any } 1 \le c \le C \text{ and } 1 \le j < n.$$

Here's an example of such an array M, where $C = 3$ and $n = 5$.

M	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$c=1$	52	6	3	3	1	1
$c=2$	55	4	2	1	1	1
$c=3$	58	4	3	3	2	1

A *study plan*, represented as a list $h = (h_1, \dots, h_C)$, describes how many hours of study you assign to each course. Each h_c is nonnegative, and $\sum_{c=1}^{C} h_c = n$. The marks you get if you use study plan h is

$$\begin{aligned} \text{marks}(h) &= \sum_{c=1}^{C} (M[c][0] + \ldots + M[c][h_c]) \\ &= \sum_{c=1}^{C} \sum_{j=0}^{h_c} M[c][j] \qquad \text{(if you prefer the sum written formally)} \end{aligned}$$

You want an *optimal* study plan, i.e., one that maximizes the sum of your course marks. For our example above, the study plan $h_1 = 2$, $h_2 = 2$, $h_3 = 1$ (i.e., 2 hours of study for course 1, 2 hours for course 2, and 1 hour for course 3) results in a suboptimal total of 184 marks:

$$(52 + 6 + 3) + (55 + 4 + 2) + (58 + 4) = 184.$$

1. (2 marks) For the above example (with $C = 3$ and $n = 5$), write down *two* optimal study plans. For each study plan, you should specify h_1, h_2, and h_3. You do not need to justify your answer.

2. (5 marks) Complete the following greedy algorithm so that it finds an optimal study study plan. A one-line description in the blank space below suffices.

function STUDYPLAN($M[1..C][0..n]$)
  initially set h_c to 0, for $1 \le c \le C$ ▷ no hours allocated yet
  for j from 1 to n **do** ▷ decide how to allocate the jth hour of study
    ▷ **describe here how to choose a course** c **that gets an additional hour**

    $h_c \leftarrow h_c + 1$ ▷ allocate another hour to course c
  end for
  return $h = (h_1, h_2, \ldots, h_C)$
end function

3. (4 marks) What is the runtime of an efficient implementation of your greedy strategy? Justify your answer, briefly explaining what data structure is useful, in light of the strategy you use to choose a course at each iteration.

4. (4 marks) Let h^G be the study plan produced by your greedy algorithm, and let h^* be an optimal study plan. Suppose that h^G and h^* are not identical. Then

   - For some course c, $h_c^G > h_c^*$ (greedy allocates more hours to course c than does h^*).
   - For some other course c', $h_{c'}^G < h_{c'}^*$ (greedy allocates fewer hours to c' than does h^*).

   Let h' be obtained from h^* by adding an hour to c, and removing an hour from c'. (So $h'_c = h_c^* + 1$, $h'_{c'} = h_{c'}^* - 1$, and $h'_i = h_i^*$ for all other courses i.) Show that

   $$\text{marks}(h') - \text{marks}(h^*) \geq 0.$$

   You can use without proof the following inequality (which should follow from your greedy strategy):

   $$M[c][h_c^* + 1] \geq M[c'][h_{c'}^*].$$

5. (4 marks) Using the previous part (and again assuming that the inequality given there holds for your greedy strategy and any optimal strategy h^*), explain why $\text{marks}(h^G) \geq \text{marks}(h^*)$.

5 Study Plan: Part II (14 marks)

In this question, we analyze **almost** the same problem as in the preceding Question 4 (Study Plan: Part I). We will use all the same definitions and notation (so go back and read Question 4 if you haven't already). **However, we make one important change: we eliminate the assumption that**

$$M[c][j] \geq M[c][j+1], \textbf{ for any } 1 \leq c \leq C \textbf{ and } 1 \leq j < n.$$

Again, we emphasize that for **this** question, you do **not** have the above assumption.

Therefore, for example, the following would be a legal array M (with $C = 3$ and $n = 5$) for **this** question, but not for Question 4:

M	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$c = 1$	52	1	1	3	3	7
$c = 2$	55	4	2	1	1	1
$c = 3$	58	4	1	3	2	3

As in Question 4, a *study plan*, represented as a list $h = (h_1, \ldots, h_C)$, describes how many hours of study you assign to each course. Each h_c is nonnegative, and $\sum_{c=1}^{C} h_c = n$. The marks you get if you use study plan h is

$$\begin{aligned} \text{marks}(h) &= \sum_{c=1}^{C} (M[c][0] + \ldots + M[c][h_c]) \\ &= \sum_{c=1}^{C} \sum_{j=0}^{h_c} M[c][j] \qquad \text{(if you prefer the sum written formally)} \end{aligned}$$

You want an *optimal* study plan, i.e., one that maximizes the sum of your course marks. For our example above, the study plan $h_1 = 2$, $h_2 = 2$, $h_3 = 1$ (i.e., 2 hours of study for course 1, 2 hours for course 2, and 1 hour for course 3) results in a suboptimal total of 177 marks:

$$(52 + 1 + 1) + (55 + 4 + 2) + (58 + 4) = 177.$$

1. (1 mark) Give an optimal study plan for the example matrix above (with $C = 3$ and $n = 5$). For your answer, you should specify h_1, h_2, and h_3. You do not need to justify your answer.

2. (3 marks) Here's a recursive algorithm to compute the value (total marks) of an optimal study plan. For the remainder of this question, we will set $C = 3$, just to make the code easier to write.

   The key idea behind the algorithm is that if you have 0 hours of study time left, then you get whatever you have earned so far; at any other point in time, you have the choice of spending one more hour on any one of the 3 courses. In each of the 3 cases, the best you can do is to get the marks for that hour, plus the best possible score for your remaining study hours after that point in time. (We will assume the matrix M and the number of hours n are global variables, since our code has a lot of parameters already.)

```
function RecursiveStudyPlan(h1, h2, h3)        ▷ h1 ≥ 0, h2 ≥ 0, h3 ≥ 0, and h1 + h2 + h3 ≤ n
    if n − h1 − h2 − h3 = 0 then                ▷ Base Case: If there are no hours left,
        a ← 0                                   ▷ add up all the marks earned.
        for c = 1 to 3 do
            for i = 0 to h_c do
                a ← a + M[c][i]
            end for
        end for
        return a
    else            ▷ v_i is the most marks we can get overall if we spend the next hour on course i
        v1 ← RecursiveStudyPlan(h1 + 1, h2, h3)
        v2 ← RecursiveStudyPlan(h1, h2 + 1, h3)
        v3 ← RecursiveStudyPlan(h1, h2, h3 + 1)
        return max(v1, v2, v3)
    end if
end function
```

   Calling RecursiveStudyPlan(0, 0, 0) computes the maximum number of marks you can earn with an optimal study plan. Note that the recursion is written somewhat strangely, with the number of hours spent going **up** on each recursive call. We coded things this way to work better with the notation we gave you. You can see that you won't get infinite recursion by noticing that the number of hours left is equal to $n - h_1 - h_2 - h_3$, which goes down by 1 in each recursive call.

   Give a tight big-O bound in terms of n on the runtime when calling RecursiveStudyPlan(0, 0, 0). You do not need to justify your answer, but doing so might help you earn partial credit.

3. (10 marks) Memoize the recursive algorithm. You should call your main function MEMOSTUDYPLAN, and it should have the same interface as RECURSIVESTUDYPLAN, i.e., it should assume that the matrix M and the number of hours n are global variables, and that you call MEMOSTUDYPLAN(0,0,0) to compute the optimal number of marks. Your algorithm will memoize solutions in a 3D table, where soln[h_1][h_2][h_3] stores the number of marks you earn when you spend h_1 hours on course 1, h_2 hours on course 2, and h_3 hours on course 3. You may also assume that your solution table is a global variable.

   You may write your code out entirely, or refer to the text of RECURSIVESTUDYPLAN with comments like "Insert lines 3–8 of RecursiveStudyPlan here."

Extra page for scratch work

Extra page for scratch work

Extra page for scratch work