THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 320: FINAL EXAMINATION – Group – April 27, 2017

Important notes about this examination

1. **You have 50 minutes to complete this exam.**
2. You are allowed up to three textbooks and (the equivalent of) a 3" 3-ring binder of notes as references. Otherwise, no notes or aids are allowed. No electronic equipment is allowed.
3. Please write darkly and clearly in non-erasable pen.
4. Good luck!

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|

Full Name:_____

Signature:_____

UBC Student #: |_|_|_|_|_|_|_|_|

7 236411 907321

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).

POSSIBLY HELPFUL NOTES
Sums of sequences:

- $\sum_{y=1}^{x} y = \frac{x(x+1)}{2}$, for $x \geq 0$.
- $\sum_{y=1}^{x} y^2 = \frac{x(x+1)(2x+1)}{6}$, for $x \geq 0$.
- $\sum_{y=0}^{x} 2^y = 2^{x+1} - 1$, for $x \geq 0$.

The Master Theorem

For a recurrence like $T(n) = aT(\frac{n}{b}) + f(n)$, where $a \geq 1$ and $b > 1$, the Master Theorem states three cases:

1. If $f(n) \in O(n^c)$ where $c < \log_b a$ then $T(n) \in \Theta(n^{\log_b a})$.
2. If for some constant $k \geq 0$, $f(n) \in \Theta(n^c (\log n)^k)$ where $c = \log_b a$, then $T(n) \in \Theta(n^c (\log n)^{k+1})$.
3. If $f(n) \in \Omega(n^c)$ where $c > \log_b a$ **and** $af(\frac{n}{b}) \leq kf(n)$ for some constant $k < 1$ and sufficiently large n, then $T(n) \in \Theta(f(n))$.

Definitions of big-O and friends:

- $f(n) \in O(g(n))$ (big-O, that is) exactly when there is a positive real constant c and positive integer n_0 such that for all integers $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) \in o(g(n))$ (little-o, that is) exactly when for all positive real constants c, there is a positive integer n_0 such that for all integers $n \geq n_0$, $f(n) \leq c \cdot g(n)$.
- $f(n) \in \Omega(g(n))$ exactly when $g(n) \in O(f(n))$.
- $f(n) \in \omega(g(n))$ exactly when $g(n) \in o(f(n))$.
- $f(n) \in \Theta(g(n))$ exactly when $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

1 What Is Your GradeScope Student #? [1 mark]

Write the GradeScope Student # of each group member clearly and legibly in the following blanks:

1. ___ ___ ___
2. ___ ___ ___
3. ___ ___ ___
4. ___ ___ ___
5. ___ ___ ___

There are a total of 75 marks available on the exam.

Make Sure That You Complete The **ONE EASY BONUS POINT QUESTION** in the group version of the exam!

2 A SAT Stand-Off [6 marks]

In class, we looked at a reduction from SAT to 3-SAT. Here we'll consider a slightly simplified reduction, from L4-SAT (a SAT variant where all clauses have four **or fewer** literals) to 3-SAT.

For clauses with 1 or 2 literals, we convert to 3-SAT clauses by repeating the last literal. We leave 3-literal clauses as they are.

To convert 4-literal clauses, we put the first three literals in one clause, and repeat the fourth literal three times in a second clause. This converts a clause of the form $(\overline{x_1} \vee x_2 \vee x_3 \vee x_4)$ to two clauses $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_4 \vee x_4 \vee x_4)$.

This reduction is **incorrect**. Complete the following to supply a counterexample to its correctness:

Counterexample instance (of L4-SAT):

Correct solution to this instance (YES or NO plus a VERY brief explanation):

Reduction's solution to this instance (YES or NO plus a VERY brief explanation):

3 $O(1)$ Answer Problems [12 marks]

1. Consider this recurrence: $T(n) = 27T(\lceil \frac{n}{3} \rceil) + \frac{\sqrt{n}}{2}$. Assume the base case is constant for small n. Fill in values below for the a, b, c, and $f(n)$ values for the Master Theorem (stated on the previous page) and the resulting asymptotic bound on the recurrence. **[3 marks]**

$a =$ $\qquad$ $f(n) =$

$b =$

$c =$ $\qquad$ bound:

2. In each blank on the left, write the letter of the Θ bound for that function. **[2 marks]**

____	$7n^2 + 12(\lg n)^3$	A. $\Theta(\lg n)$
____	$2^{(n^{0.5})}$	B. $\Theta((\lg n)^3)$
____	$\frac{n^2 \lg n}{n(n+1)}$	C. $\Theta(n^2)$
____	$2^{(n/2)}$	D. $\Theta(2^{\sqrt{n}})$
		E. $\Theta((\sqrt{2})^n)$

3. `3-SAT` is known to be NP-complete. We reduce a new decision problem `MYSTERY` to `3-SAT` (correctly and in polynomial time). Answer the following two questions about what this reduction establishes. **[3 marks]**

Does this establish that `MYSTERY` is in NP? Circle **one**.	**YES**	**NO**
Does this establish that `MYSTERY` is NP-complete? Circle **one**.	**YES**	**NO**

4. True or False. **[4 marks]**

   (a) An adversary can provide input that forces worst-case performance from Randomized QuickSort.
   TRUE $\qquad$ **FALSE**

   (b) A reduction must work for any legal input to the original problem.
   TRUE $\qquad$ **FALSE**

   (c) NP-complete problems are the hardest kinds of algorithmic problems.
   TRUE $\qquad$ **FALSE**

   (d) Assuming P $\neq$ NP, the following subproblem of the Steiner Tree Problem is in P (i.e., can be solved in polynomial time): the subproblem containing only the instances where all nodes are shaded.
   TRUE $\qquad$ **FALSE**

4 Clark Kent's Glasses [8 marks]

Consider these well-known algorithmic problems or approaches:

A.	Vertex Cover	D.	Graph Colouring	G.	Minimum spanning tree
B.	Hamiltonian Cycle	E.	Traveling Salesman	H.	PageRank
C.	Subset Sum	F.	Bloom Filters	I.	Sorting

Each of the problems below is a "disguised" version of one of the problems above. Some problems above may be used multiple times; others may not be used at all. For each problem below, write the letter of the problem above it best matches. **[2 marks per question]**

1. _____ You're writing a solver for numeric puzzle problems. The puzzle has a number of blanks in it, where each blank needs to be filled with a single digit 0–9. Certain pairs of blanks have to have different digits from each other (the "constraints"). You want to determine if it's possible to fill a digit into each blank without violating any constraints.

2. _____ You're designing an assembly line to make circuit boards. Each circuit board needs a large number of holes drilled into it at specific locations, in any order. You have a single drill head, which drills a hole and then moves to the location of the next hole to be drilled. After all holes have been drilled, it has to move back to its starting position to repeat the process for the next circuit board. You want to figure out the fastest way to drill the holes.

3. _____ You want your mobile app to include a basic spellcheck feature that underlines words that aren't in a large set of foreign language dictionaries. But you don't have enough memory space available to store all the dictionaries. You decide that you're okay with your spellchecker making a mistake on no more than 0.5% of words as long as you can substantially reduce the amount of memory you have to use.

4. _____ You're an animal researcher, and you've designed a maze for rats that consists of several chambers connected by tunnels. You're interested in knowing whether the rats move randomly through the maze. So you want to know: if they *do* traverse the maze at random, how often will a rat end up in each chamber?

5 eXtreme True And/Or False [8 marks]

Each of the following statements may be **always** true, **sometimes** true, or **never** true. Select the best of these three choices and then:

- If the statement is **always** true, (1) give and very briefly explain an example instance in which it is true and (2) sketch the key points of a proof that it is always true.
- If the statement is **never** true, (1) give and very briefly explain an example instance in which it is false and (2) sketch the key points of a proof that it is never true.
- If the statement is **sometimes** true, (1) give and very briefly explain an example instance in which it is true and (2) give and very briefly explain an example instance in which it is false.

Note that you will always select a choice and then give two answers. There is space for this below.

1. In a connected graph with $n \geq 3$ nodes, at most $n-1$ nodes are diametric. (Recall: the *diameter* of a graph is the maximum over any pair of nodes of the shortest distance between that pair of nodes. Further, if the shortest path from i to j in a graph is the diameter, we say that each of i and j is *diametric*.) **[4 marks]**

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

2. A DAG (directed, acyclic graph) with at least two nodes is strongly connected. **NOTE:** for this problem, we assume that a *strongly connected* directed graph is one in which each node is reachable from each other node besides itself. Thus, any DAG with zero or one nodes **is** strongly connected. **[4 marks]**

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

6 A Stable of Matching Problems [6 marks]

Complete the reduction below from a variant of the stable matching problem called FMP to the weighted bipartite matching problem (WBM). In FMP, instead of avoiding instabilities, we want to maximize the number of people who end up with their first choice of partner. (Further, we still require that everyone ends up married.)

At the bottom of the page is a full description of the weighted bipartite matching problem. **[6 marks]**

REDUCTION: Given an instance of FMP with n men $m_1, m_2, \ldots, m_n$ and n women $w_1, w_2, \ldots, w_n$, create a weighted bipartite graph, where

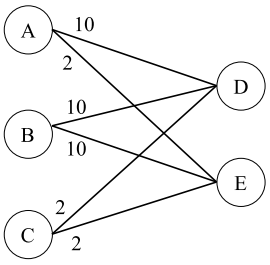
1. There are _____ vertices on the **left** side of the graph and _____ vertices on the **right**.
2. Each vertex on the **left** side of the graph v_{L_i} represents _____.
3. Each vertex on the **right** side of the graph v_{R_i} represents _____.
4. There is an edge (v_{L_i}, v_{R_j}) between each pair of a vertex on the left and a vertex on the right.
5. The weight of edge (v_{L_i}, v_{R_j}) is: _____

   _____.

Now, solve the WBM instance and then produce the FMP instance's solution by converting each edge (v_{L_i}, v_{R_j}) in the WBM maximum matching into the married couple _____.

For reference: In the **Weighted Bipartite Matching** problem, you are given a weighted bipartite graph $G = (V_1, V_2, E)$, and you compute a maximal matching $E' \subseteq E$. A *matching* is a set of edges such that no two edges share a vertex (i.e., no vertex "appears" more than once in the matching), and a *maximal matching* is a matching of maximum weight, where the weight of a matching is the sum of the weights of its edges.

For example, in the graph to the right, the maximal matching is the edges (A, D) and (B, E) with a weight of $10 + 10 = 20$.



7 The Problem, InAWord [15 marks]

Given a string of letters, you want to determine whether there's a way to insert spaces so that the string becomes a sequence of valid English words. For example, the string `mysunflower` works because putting a space after letter 2 produces `my sunflower`, or putting spaces after letters 2 and 5 produces `my sun flower`. The string `abcdef` does not work.

Assume you have a helper function `isWord(str)` that runs in constant time and returns `True` if `str` is an English word, and `False` otherwise.

1. A friend proposes this greedy approach: Starting at the beginning of the string, add one new character at a time, and check if the characters form a word. As soon as you find a word, put a space there, and recursively call the same algorithm on the rest of the string. Here is pseudocode:

```
function canSplit(str):
    for i = 1 to length(str)-1
         if isWord(str[1 .. i]) // are the first i letters a word?
              return canSplit(str[i+1 .. length(str)])
    // At end of loop: no prefix is a word; so, we return
    // True if and only if the entire string is a word.
    return isWord(str)
```

   (a) Assume that `isWord` returns `True` for the strings `a`, `at`, `ate`, `tea`, `cup`, and `teacup`, and `False` for all other strings. In that case, the greedy algorithm returns `True` for the input string `ateacup`. How does it split `ateacup` as it runs? Circle the **best** answer. **[1 mark]**

   i. `a teacup`
   ii. `a tea cup`
   iii. `ate a cup`
   iv. The function finds a different splitting.

   (b) Complete the following to supply a counterexample that proves this greedy approach is **not** correct. **Note:** this counterexample uses made-up words rather than English ones. **[2 marks]**

   List of ALL strings that should be considered words: `a`, `aa`, `bb`, `bbb`.

   Counterexample instance:

   Correct solution to this instance (including an actual split, if the answer is True):

   Greedy solution to this instance (including an actual split, if the answer is True):

2. Complete the following recurrence to solve the `canSplit` problem for non-empty strings in such a way that we require only linear table entries (in the length of `str`) to convert this to dynamic programming. **[6 marks]**

$$\texttt{canSplit(str)} = \begin{cases} \texttt{True} & \text{if } \texttt{isWord(str)} \text{ is true} \\ \exists i \in \text{the integers 1 up to (and including) } \_\_\_\_\_ \text{ such that:} & \\ \quad \_\_\_\_\_ & \text{otherwise} \\ \quad \_\_\_\_\_ & \end{cases}$$

3. Complete the following dynamic programming pseudocode function to solve the `canSplit` problem based on the recurrence above. **[6 marks]**

```
function canSplit(str):
    n = length(str)

    // Set up the array:
    let Table be an array with indexes 1 .. __________        // FILL IN THE BLANK

    for i = ____________________________:                     // FILL IN THE BLANK

        if ___________________:                               // FILL IN THE BLANK

                                                          // FILL IN THE "THEN" CASE




        else:

                                                          // FILL IN THE "ELSE" CASE









    return ________________                                   // FILL IN THE BLANK
```

8 Majority Sortition [11 marks]

You're given a **sorted** array of integers and asked if a given element x is a *majority element* in the array. That is, in an array of size n, we want to know if the number of appearances of element x is $> \frac{n}{2}$. (This is equivalent to the number of appearences being $\geq \lfloor \frac{n}{2} \rfloor + 1$.)

1. Consider the following two example arrays and circle the best answer for the question about each one. **[2 marks]**

   (a) We list only the first four elements of the array below. What do we know about the value of `A[7]` if the element `25` is a majority element?

`index`	1	2	3	4	5	6	7	8	9
`A[index]`	15	15	18	25			?		

   Circle the **best** answer: `A[7] = 25` `A[7]` $\neq$ `25` we know **NEITHER**

   (b) We list only the first six elements of the array below. Is 25 is a majority element?

`index`	1	2	3	4	5	6	7	8	9
`A[index]`	15	15	16	16	16	25			

   Circle the **best** answer: **YES** **NO** **UNKNOWN**

CONTINUED ON THE NEXT PAGE

2. Fill in the following $O(\log n)$ time algorithm to solve this problem. (Note that there are two functions, and both include portions to be filled in.) **[9 marks]**

```
function isMajority(A, x):
    // Determine whether x is a majority element in A.
    // A has indexes 1, 2, 3, .., n

    n = length(A)
    i = firstOccurrence(A, x, 1, n)

    if i == -1:
         return False

    else if ________________________________________________:

         return True
    else:
         return False

function firstOccurrence(A, x, low, high):
    // returns index of first occurrence of x in A[low..high]

    if ____________________:
         // subarray A[low .. high] is empty
         return -1

    mid = floor((low+high)/2)

    if ______________________________:
         // The first occurrence is at index low
         return low

    else if mid > low and ______________________________________:
         // The first occurrence is at index mid.
         return mid

    else if A[mid] < x:
         // The first occurrence is in the right half.

         return ___________________________________

    else:
         // The first occurrence is in the left half.

         return ___________________________________
```

9 Fog-o-War [8 marks]

RECALL the **TUG-O-WAR** problem: given n people $p_1, \ldots, p_n$ (for $n > 1$) and their weights $w_1, \ldots, w_n$, we want to generate two teams that are well-balanced.

Specifically, we consider this decision variant of the problem: given n people with their weights and a target difference k, can we assign each of the n people into one of two teams (team 1 and team 2) such that the absolute difference in total weight between team 1 and team 2 is less than or equal to k?

1. Give a good certificate for TUG-O-WAR problem. (Note: you do not need to prove that TUG-O-WAR is in NP, but your certificate should be a good choice for such a proof.) **[2 marks]**

2. Recall the SUBSET-SUM problem: given a list of numbers $A = a_1, \ldots, a_n$ and a value W, does there exist a subset of elements of A that sum to W?

   SUBSET-SUM is known to be NP-complete. Give a reduction from SUBSET-SUM to TUG-O-WAR to prove that TUG-O-WAR is NP-hard. Do not prove the correctness of your reduction, but you **should** clearly explain the key elements of your reduction and why they are there.

   Assume that all elements of A are positive, and that $\frac{m}{2} \leq W \leq m$, where m is the sum of all elements in A. **Remember** that your algorithm creates the TUG-O-WAR instance and can therefore, e.g., freely choose the value of k to be a simple and useful one. **[6 marks]**

10 Bonus [3 BONUS marks]

THE NEW FIRST QUESTION IS EASY: COMPLETE IT!

Bonus marks add to your exam and course bonus mark total but are **not** required. **WARNING**: These questions are too hard for their point values. We are free to mark these questions harshly. Finish the rest of the exam before attempting these questions. Do not **taunt** these questions.

1. What was your favorite algorithm, analysis technique, proof, or other part of the course? (Worth 1 bonus point.)

   NOTE: Write something quick now (for full credit) and then come back later in the unlikely event that you have time and write a considered answer (to be a hero like Squirrel Girl).

2. Recall the majority problem described earlier in this exam. Give and briefly justify the correctness of an algorithm `unsortedMajority(A)` that runs in $O(n)$ time, takes as input an **unsorted** array of numbers `A` of length n, and produces the majority element if there is one and otherwise any arbitrary element of the array.

3. **Answer Like an "eXtreme True And/Or False" Question:** In a connected graph with $n \geq 3$ nodes, at most $n - 2$ nodes are articulation points. (Recall: an *articulation point* is a node whose removal increases the number of connected components in the graph.)

   Circle **one**: **ALWAYS** **SOMETIMES** **NEVER**

   (1)

   (2)

4. Give a clear and correct reduction from `dHAMCYCLE` (the directed Hamiltonian Cycle problem) to `uHAMCYCLE` (the undirected Hamiltonian Cycle problem). You **must** clearly and concisely justify the correctness of your reduction but need not formally prove its correctness.

   (Reminder: the Hamiltonian Cycle problem asks whether there is a simple cycle in a graph that includes every node in the graph. You may assume that the graph is guaranteed to have at least three vertices.)

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.

This page intentionally left (almost) blank.
If you write answers here, you must CLEARLY indicate on this page what question they belong with AND on the problem's page that you have answers here.