THE UNIVERSITY OF BRITISH COLUMBIA
CPSC 320: MIDTERM #1 – Individual – October 10, 2018

Important notes about this examination

1. **You have 65 minutes to complete the 5 problems on this exam worth a total of 40 marks.**
2. You are allowed up to two sides of a letter-size sheet of notes as references. Otherwise no notes or aids are allowed. No electronic equipment is allowed.
3. Good luck!

Full Name:_____

Signature:_____

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

Enter as your exam ID the rightmost 3 digits of the number printed in the corner of each page of your exam:

Exam ID: |_|_|_|

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).

WRITE UGRAD IDs HERE (**-1 mark** if any missing or incorrect; use only the boxes you need)

UGRAD ID:

NOTE: Throughout the exam a *simple* graph is an undirected, unweighted graph with no multiple edges (i.e., no exact repeats of the same edge) and no self-loops (i.e., no edges from a vertex to itself). Graphs are simple unless stated otherwise, and even where we explicitly contradict one of these, the rest remain true. So, for example, a "directed graph" with no other information specified would be unweighted with no multiple edges and no self-loops.

1 Asymptotically yours

Here are two statements about non-negative functions $f(n)$ and $g(n)$ over the positive integers. For each, indicate whether it is

- **Always** true for every possible pair of non-negative functions $f(n)$ and $g(n)$.
- **Sometimes** true for pairs of non-negative functions $f(n)$ and $g(n)$ (and false for other pairs $f(n)$, $g(n)$).
- **Never** true for any possible pair of non-negative functions $f(n)$ and $g(n)$.

Here are the statements:

1. If $f(n) = cg(n) + o(g(n))$ for some constant c then $f(n) = O(g(n))$.
   ◯ Always ◯ Sometimes ◯ Never
2. If $\log f(n) = O(\log g(n))$ then $f(n) = O(g(n))$.
   ◯ Always ◯ Sometimes ◯ Never

Now, give examples of nonnegative functions $f(n)$ and $g(n)$ satisfying the following relationships (you should use a different example for each relationship):

3. $f(n) \neq g(n)$ but $f(n) = g(n) + h(n)$ where $h(n) = o(g(n))$.

   $f(n) =$ $g(n) =$

4. $f(n) = O(g(n))$ but $f(n) \neq \Theta(g(n))$.

   $f(n) =$ $g(n) =$

5. $f(n) = O(g(n))$ but $2^{f(n)} \neq O(2^{g(n)})$.

   $f(n) =$ $g(n) =$

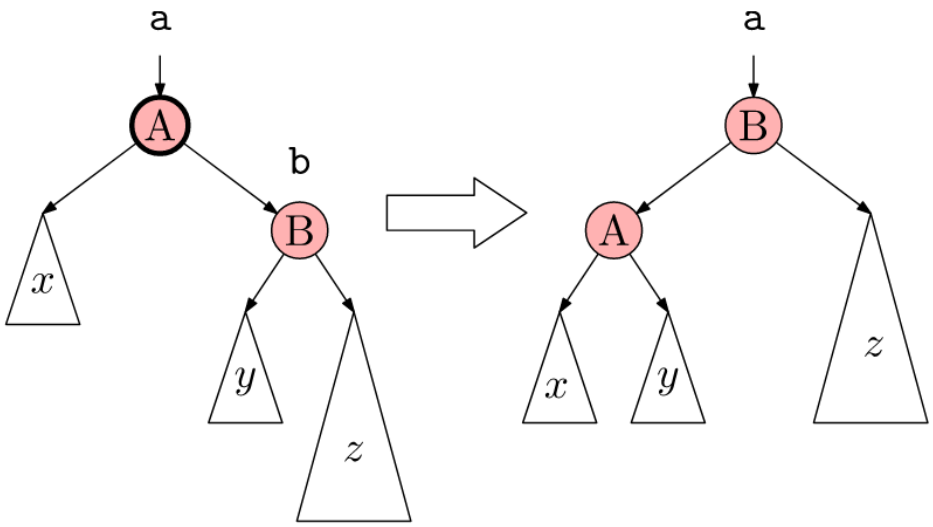
2 Knowing your trees

1. Let T be a binary search tree with n keys. In Assignment #1, we considered storing in each node N of the tree the size of the subtree rooted at N, and you described an algorithm that uses this information to count in $O(\log n)$ time the number of keys x of T such that $k_1 < x < k_2$ for some given parameters k_1 and k_2.

   Using the size information stored in each node is only half the job, however. Write appropriate code in each of the blanks in the following implementation of `insert_key` to keep the size information accurate when nodes are inserted in T. Assume that the size field of a node `node` is stored in `node.size`. Note that some of the blanks can (and should) be left blank, and the code does **not** have to maintain balance in the tree.

```
define insert_key(start_node, new_key)

    [                                        ]

    if start_node == null:

        [                                    ]

        newnode = new node(key)

        [                                    ]

        return newnode

    if new_key < start_node.key():

        [                                    ]

        start_node.left_child = insert_key(start_node.left_child, new_key)

    else if new_key > start_node.key():

        [                                    ]

        start_node.right_child = insert_key(start_node.right_child, new_key)

    return start_node
```

2. Now consider a self-balancing AVL tree. Describe how to update the size fields of nodes A and B after a single rotation as illustrated in the figure is performed (we will spare you the other types of rotations). Note that a refers to the parent of the node containing key A, and that x, y and z represent subtrees. Ignore the label b that appears in the picture; it plays no role in this question.



A.size =

B.size =

3 More exchange of kidneys

Recall that an instance of the Kidney Exchange Problem (KEP) is (n,M,R), where n is the number of patients (and also the number of donors), M is an initial perfect matching of patients and donors, and R is a collection of complete rankings R[p] for each of the n patients p. The donor d initially matched with p may not be at the top of p's ranking.

For an instance I of KEP, the **best-preference graph** G_I has one node per pair (d, p) in the initial matching M. There is a directed edge from (d, p) to (d', p') if d' is the most highly ranked donor on p's preference list.

The *Exchange* algorithm of Assignment 2 for matching patients and donors ignores the reality that for large n, it is impractical to get n pairs of patients and donors in the same hospital and do all the exchanges (surgeries) at the same time. In practice, exchanges typically involve just two pairs.

Recall that for a given instance I of KEP, a valid solution S is a perfect matching between patients and donors. S is a *2-exchange* if the following holds for all pairs of patients p and p': If (p, d) and (p', d') are in the initial matching M and (p, d') is in S, then (p', d) is also in S.

We say that 2-exchange S is *stable* if there do not exist distinct pairs (d, p) and (d', p') such that patient p prefers d' to d **and also** patient p' prefers d to d'.

1. Show an instance of KEP with $n = 3$ for which there is no stable 2-exchange.

2. Is the number of 2-exchanges upper bounded by a polynomial in n, for all instances I with n patients?
   ◯ Yes ◯ No

3. Are there instances I with n patients for which the number of 2-exchanges exponential in n, for sufficiently large n?
   ◯ Yes ◯ No

4. Let *2Exchange* be the following algorithm (a variant of the *Exchange* algorithm from Assignment 2), which arranges exchanges involving two patient-donor pairs, as well as degenerate exchanges involving just one patient-donor pair. The algorithm never arranges exchanges involving three or more pairs.

```
Algorithm 2Exchange (I = (n,M,R))
    Initialize S to be the empty set
    While n > 0
        Create the best-preference graph G_I for instance I
        If G_I has a directed cycle, say v_1, v_2, ..., v_k = v_1, for k = 2 or k = 3 then
            Let v_i be the pair (d_i, p_i), for 1 ≤ i < k
            Update instance I as follows:
                Remove the pairs (d_i, p_i) from M, 1 ≤ i < k
                Remove d_i from all patient rankings in R, 1 ≤ i < k
                Set n to n − k + 1
            Add the pairs (d_{i+1}, p_i) to S for 1 ≤ i < k − 1, and also add the pair (d_1, p_{k−1}) to S
        Else
            Remove all remaining pairs from M and add them to S
            Set n to 0
    Endwhile
    Return S
```

Is the output of Algorithm *2Exchange* always the same on any instance, no matter which directed cycle (with $k = 2$ or $k = 3$) is chosen at each iteration of the While loop?

◯ Yes ◯ No

5. Does Algorithm *2Exchange* always produce a 2-stable solution on instances I that have a 2-stable solution?

◯ Yes ◯ No

4 More on graphs

In this problem, let G be a simple, connected graph with at least two nodes. By the *depth* of a rooted tree, we mean the number of edges of the longest path from the root to a leaf.

For each of the following statements about graph G, indicate whether the statement is:

- Always true for every simple connected graph G with at least two nodes
- Sometimes true for a simple connected graph G, but not for other simple graphs G with at least two nodes
- Never true for any simple graph G with at least two nodes.

Here are the statements:

1. Two different depth first search (dfs) trees rooted at the same node s of G have the same depth.

◯ Always ◯ Sometimes ◯ Never

2. Two different dfs trees rooted at two *different* nodes s and s' of G have the same depth.

◯ Always ◯ Sometimes ◯ Never

3. Two different breadth first search (bfs) trees rooted at the same node s of G have the same depth.

◯ Always ◯ Sometimes ◯ Never

4. Two different bfs trees rooted at two *different* nodes s and s' of G have the same depth.

◯ Always ◯ Sometimes ◯ Never

5 Triathlon reductions

The *Triathlon-stable matching* problem is defined as follows. An instance involves $3n$ athletes: n swimmers, n runners and n cyclists. Each athlete in one group has a preference list for the athletes in each of the other two groups (that is, a runner will have a preference list for the swimmers, and another separate preference list for the cyclists, and so on). You want to find n triples (groups of three), with one swimmer, one runner and one cyclist per triple, to compete in a triathlon. Ideally your solution of n triples should have no instabilities. The first part of this problem asks you to come up with a reasonable notion of instability.

1. Complete the following definition of instability applied to this new situation. With respect to a solution S, an instability consists of two triples (s, r, c) and (s', r', c') of S such that

   Your answer goes here.

2. Let A be an algorithm that, given the preference lists of all athletes, always finds solution with no instabilities, if the instance has such a solution. Complete the following reduction that uses A to solve the regular stable matching problem:

   Given an instance I of [ ] we transform it into an instance of [ ] as follows:

   The rest of your reduction goes here.