THE UNIVERSITY OF BRITISH COLUMBIA

CPSC 221: FINAL EXAMINATION – June 27, 2019

Full Name: ______________________ CS Account: _____________

Signature: ______________________ UBC Student #: ________

Important notes about this examination

1. You have 150 minutes to complete this exam.
2. No assistance of any kind is allowed (or needed).
3. You must submit all pages of this booklet, including any detached pages.
4. In case of uncertainty, clearly write down your assumptions.
5. Good luck!

Please do not write in this space:

1 236413 048929

CPSC 221 2019S: Final Exam

June 27, 2019

1 Who gets the marks? [1 marks]

Please enter your 4 or 5 digit CSID in this box: $\boxed{}$

2 Tree Shorts [12 marks]

(a) Suppose you do post-, pre-, in- and level-order traversals of an arbitrary Binary Search Tree with more than one node. Which of the following statements is (are) true about the traversals?

- ○ Only the running time of level-order traversal depends on the height of the tree.
- ○ The largest value in the tree is output last in exactly 2 of the traversals.
- ○ There exists a tree for which at least 2 of the traversals output the nodes in increasing order.

(b) Suppose a friend claims his/her AVL Tree `remove` function runs in time $O(\log(n^3))$. After thinking about it you have some concerns. Which of the following observations are valid?

- ○ The algorithm is inefficient because it is asymptotically slower than $O(\log(n))$.
- ○ This running time only reflects average running time, not worst case.
- ○ This running time neglects to account for rotations.

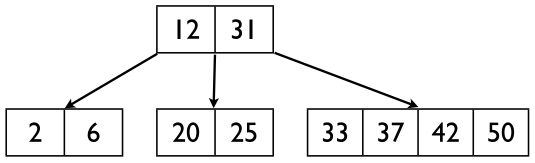
(c) Suppose that you have a binary tree of height h containing n nodes. What is the running time of an efficient algorithm to determine if the tree is in order. (i.e at every node k, all the elements in the k's left subtree are smaller or equal to k's key and all the elements in the k's right subtree are larger or equal to k's key.)

○ $\Theta(1)$ ○ $\Theta(h)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$

(d) Suppose that, for every node v in a binary tree of height h containing n nodes, you wish to compute the length of the longest path from v down to a leaf, storing that distance in the node. What is the running time of the best algorithm to do this?

○ $\Theta(1)$ ○ $\Theta(h)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$

(e) Consider the BTree below, and assume that the root node has already been loaded into main memory. How many disk seeks are necessary in a search for key 37?



○ 0 ○ 1 ○ 2 ○ 3 ○ 4 ○ more than 4

(f) Suppose that you have a binary tree of height h containing n nodes. What is the running time of an efficient algorithm to print the keys in order by key (alphabetically, for example).

○ $\Theta(1)$ ○ $\Theta(h)$ ○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$

3 Hash Shorts [16 marks]

(a) **[6 marks]** The table on the left gives a hash function for a set of keys. The keys are inserted into an originally empty hash table in some order, using linear probing to handle collisions. The state of the hash table after the insertions is illustrated below the hash function.

key k	hash $h(k)$
A	2
O	5
H	2
C	2
G	1
B	0
E	0

0	1	2	3	4	5	6
B	E	A	C	H	G	O

In the table below, fill in the bubbles corresponding to the keys that satisfy each situation.

	A	O	G	B
Could have been the first key entered:	○	○	○	○
Could have been the last key entered:	○	○	○	○
Must have been entered before E:	○	○	○	○
Must have been entered after E:	○	○	○	○

(b) **[2 marks]** Give an expression for the *load factor* for a hash table of size s containing t keys:

(c) **[2 marks]** Which of the following statements describes a *collision* in a hash table?

- ○ Two entries are identical, except for their keys.
- ○ Two entries with different data have the exact same key.
- ○ Two entries with different keys have the exact same hash value.
- ○ Two entries with different hash values have the exact same key.

(d) **[4 marks]** Suppose that your hash function does not satisfy the simple uniform hashing assumption (SUHA). Which of the following can result?

- ○ Poor performance for `insert(k)`.
- ○ Poor performance for a successful `find(k)`.
- ○ The time it takes to compute the hash function increases with every new insertion.
- ○ Uneven distribution of chain lengths in a separate-chaining hash table.
- ○ Large clusters could form in a linear-probing hash table.
- ○ The same key may hash to two different indices.
- ○ A separate-chaining hash table can become 100% full.

(e) **[2 marks]** Suppose a hash table has size 10, and that the keys are 64 bit unsigned binary integers. We want to hash 7 unknown values from this keyspace.

$$h(k) = (\texttt{product of } k'\texttt{s bits})^2 \mod 10$$

Which of these ideal hash function characteristics is/are violated by this hash function?

- ○ A good hash function distributes the keys uniformly over the array.
- ○ A good hash function is deterministic.
- ○ A good hash function is computed in constant time.

4 Heap Shorts [12 marks]

(a) **[4 marks]** Suppose you are given a min-heap containing n keys, implemented using a 1-based indexing strategy. Which of the following statements are true?

○ Calling BuildHeap on the heap results in no change to the structure.

○ Calling BuildHeap on the heap runs in $O(1)$ time.

○ Reversing the contents of the array produces a max-heap.

○ Insertion sort will fully sort the array contents in $O(n)$ time.

○ The heap order property is not violated after running Merge sort on the array contents (in place).

○ If the n keys in the array `arr` are the integers $1 \ldots n$, then 1 is in `arr[1]`, 2 is in either `arr[2]` or `arr[3]`, and 3 is in either `arr[2]` or `arr[3]`.

○ Suppose that instead of a heap, you are given an array `arr` containing an increasing sequence of n keys in locations `arr[0]...arr[n-1]`. You can adjust this array into a 1-based min-heap in logarithmic time.

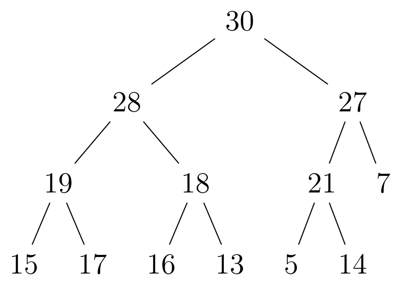
(b) **[2 marks]** Suppose you build an array-based heap by repeated insertion into the structure, and that the array is resized k times over the sequence of insertions. How many entries are in the heap?

○ $\Theta(1)$ ○ $\Theta(\log k)$ ○ $\Theta(k)$ ○ $\Theta(k \log k)$ ○ $\Theta(2^k)$

(c) **[2 marks]** Consider an unordered array containing n elements. Suppose we turn it into a heap by an algorithm similar to BuildHeap, except that we start from the last element, calling HeapifyDown on *every* index in the array. What is the complexity of this adapted algorithm?

○ $\Theta(\log n)$ ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$ ○ $\Theta(2^n)$

(d) **[4 marks]** The tree below is a `maxHeap`, defined analogously to the `minHeap`s we have studied in class, except that every path from a root to a leaf is non-increasing. Use this heap to answer the following 2 questions:



(a) **[2 marks]** The last operation was an insert of a key x. Circle the possible values of x.

5 7 13 14 15 16 17 18 19 21 27 28 30

(b) **[2 marks]** A call to `removeMax` changes the position of which of the following keys?

5 7 13 14 15 16 17 18 19 21 27 28 30

5 Graph Shorts [15 marks]

(a) **[7 marks]** A simple, connected, undirected graph contains 2019 *edges*.

i. Give the greatest lower bound for the number of *vertices* in the graph: $\boxed{}$

ii. Give the least upper bound for the number of *vertices* in the graph: $\boxed{}$

iii. Is the graph complete? ◯ Yes ◯ No ◯ Maybe

iv. Suppose the graph has n vertices, then breadth first search, when run on the graph, labels $\boxed{}$ *cross* edges.

(b) **[2 marks]** Select each property that **by itself** ensures that breadth-first search and depth-first search label the same set of discovery edges when run on a simple, undirected graph G.
◯ G is acyclic. ◯ the vertices of G have degree 2. ◯ G is a tree. ◯ G is connected.

(c) **[2 marks]** Suppose we have run Dijkstra's algorithm on a graph G to find a shortest path from vertex u to vertex v. Call that path P.

i. If we increase the weight of every edge in G by a constant amount, is path P still a shortest path from u to v? ◯ Yes ◯ No

ii. If we divide the weight of every edge in G by 2, is P still a shortest path from u to v?
◯ Yes ◯ No

(d) **[2 marks]** Suppose we have found a minimum spanning tree (MST), T, by executing Kruskal's algorithm on graph G.

i. If we increase the weight of every edge in G by a constant amount, is tree T still a MST of G?
◯ Yes ◯ No

ii. If we divide the weight of every edge in G by 2, is tree T still a MST of G? ◯ Yes ◯ No

(e) **[2 marks]** Given an adjacency list representation of a directed graph, we would like to reverse all the edges in the graph to create a new graph. Here's the code to do it, but one line is missing. Select the correct line of code to fill in the blank.

```
struct LNode {
  // constructor initializes vtx and next.
  LNode( int v, LNode *n ): vtx(v), next(n) { }
  int vtx;
  LNode *next;
};
vector<LNode *> reverseEdges( vector<LNode *> G ) {
  vector<LNode *> H(G.size(), NULL);
  for( unsigned int i=0; i < G.size(); i++ ) {
    LNode *x = G[i];
    while( x != NULL ) {
                     //<-- Missing line goes here.
      x = x->next;
    }
  }
  return H;
}
```

◯ `H[x->vtx] = new LNode(i, H[x->vtx]);`

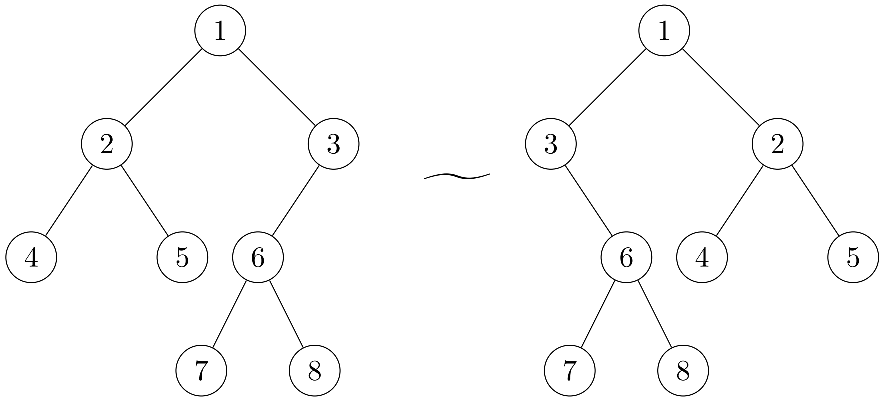
◯ `H[x->vtx] = new LNode(x->vtx, G[i]);`

◯ `G[x->vtx] = new LNode(x->vtx, H[i]);`

◯ `G[x->vtx] = new LNode(i, G[x->vtx]);`

6 Swizzle [12 marks]

A binary tree is a *swizzle* of another if it can be obtained from the other by a series of *swizzle-swap* operations. A *swizzle-swap* at a node simply swaps the position of its left and right children. In a valid swizzle, any number of nodes can be swizzle-swapped. Two empty trees are swizzles of one another. The pair of trees below are swizzles: 1's children have been swapped, and 3's children have been swapped.



(a) **[6 marks]** Answer the following questions about swizzles!

i. ☐ (T or F) Swizzle-swaps can be used to change an arbitrary binary tree into a binary search tree.

ii. ☐ (T or F) A swizzle of a heap containing $2^{h+1} - 1$ nodes (where h is the heap's height), is also a heap.

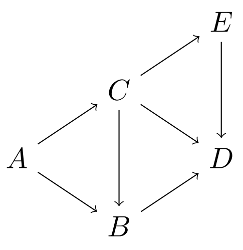
iii. ☐ (T or F) Tree height is invariant under the swizzle-swap operation.

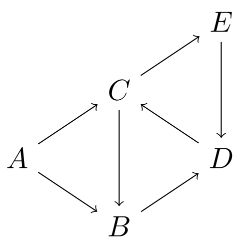
(b) **[6 marks]** Complete the code below for a function `bool swizzle(node * t1, node * t2)`, which returns `true` if `t1` and `t2` are the roots of trees which are swizzles of one another, and `false` otherwise. Assume that a `node` is a class containing integer valued `data`, and node pointers `left`, and `right`, and that all of these members are public. Comment your code so that every program structure (conditional cases, loops, etc.) has an easily identifiable purpose.

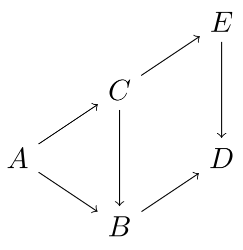
```
bool swizzle(node * t1, node * t2){

if (t1==__________ && t2==__________) return __________;

else if (t1!=_________ && t2!=_________)

    return ________________________ &&

        ________________________________________________________

        ________________________________________________________;

    else return __________;
}
```

7 Graph Games [15 marks]

(a) **[5 marks]** Circle the graph(s) for which `A C B E D` is a valid topological sort. If that ordering is not unique, give another valid ordering in the space below the graph(s).

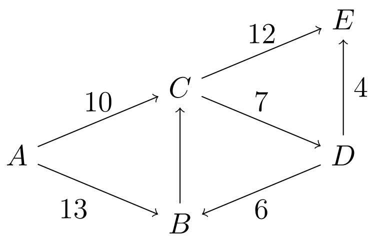






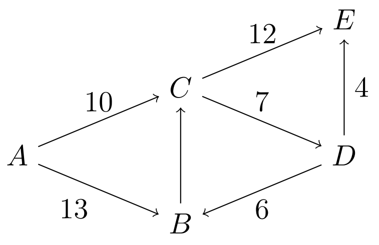
(b) **[5 marks]** For each graph below, state the largest weight for directed edge (B, C) that makes the given description true.

A shortest path from `A` to `E` does not exist.



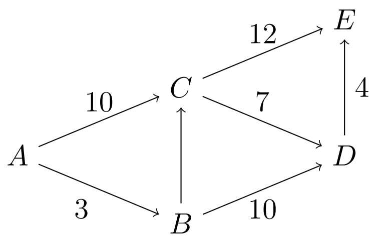
$weight(B, C) =$

A shortest path from `A` to `E` exists, but Dijkstra's algorithm will not find it.



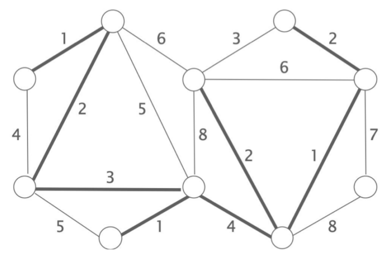
$weight(B, C) =$

The unique shortest path from `A` to `E` is `A B C D E`.

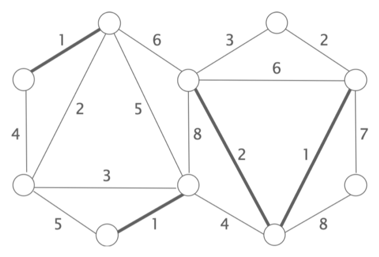


$weight(B, C) =$

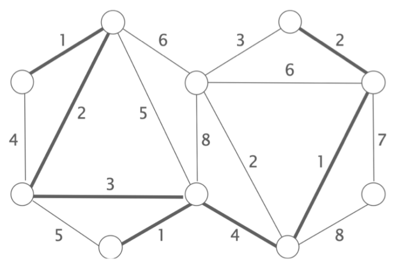
(c) **[5 marks]** In each of the graphs below, we have highlighted a subset of the edges. Determine whether the chosen edges could possibly represent a partial spanning tree obtained from (a prematurely stopped) Prim's algorithm, (a prematurely stopped) Kruskal's algorithm, both, or neither. Fill in the blanks with your choices.



○ Prim's ○ Kruskal's



○ Prim's ○ Kruskal's



○ Prim's ○ Kruskal's

8 Running Buddies [12 marks]

Suppose your BFF shows up to your place with some summer treats – fancy new versions of familiar functions. In each problem below, 1) decide whether or not the new function improves the worst case running time of the given algorithm (if it doesn't help, don't use it), and 2) based on your answer, describe the running time. In each case, n represents the size of the problem–either the number of elements for the sorting algorithms, or the number of vertices for the graph algorithms. Be sure to give the tightest bounds you can. Note: for the graph algorithms, you may assume the graph is simple and connected, but you know nothing else about the number of edges (and you should assume the worst case).

(a) What is the running time of *Selection Sort* if your BFF gives you a *findMin* function whose running time is $O(1)$?

Do you use the function? ○ Yes ○ No Based on this choice, the running time is $\Theta(\boxed{})$.

(b) What is the running time of *HeapSort* if your BFF gives you a priority queue whose functions run in $O(1)$ time?

Do you use the function? ○ Yes ○ No Based on this choice, the running time is $\Theta(\boxed{})$.

(c) What is the running time of *Prim's* Minimum Spanning Tree algorithm if your BFF gives you a priority queue whose functions run in $O(1)$ time?

Do you use the function? ○ Yes ○ No Based on this choice, the running time is $\Theta(\boxed{})$.

(d) What is the running time of *BFS* if your BFF gives you a queue whose functions run in time $\Omega(\log n)$?

Do you use the function? ○ Yes ○ No Based on this choice, the running time is $\Theta(\boxed{})$.

(e) What is the running time of *Kruskal's* Minimum Spanning Tree algorithm if your BFF gives you a Disjoint Sets structure whose functions run in $O(1)$ time?

Do you use the function? ○ Yes ○ No Based on this choice, the running time is $\Theta(\boxed{})$.

9 Road Trip [10 marks]

It's time to plan your summer road trip! Suppose you want to travel from UBC to a music festival in Atlin, BC in your fancy new RV, but you're not sure if the roads are wide enough. Luckily, you have a width-map of BC where all the roads are marked with the maximum width vehicle they will allow. Design an algorithm that takes a weighted graph representing the width-map, and the width of your RV as input, and that returns a route between UBC and Atlin whose minimum width is at least the width of your RV, if such a route exists, and FAIL, otherwise.

(a) **[2 marks]** The route can be found most efficiently by a modification of which of the following algorithms? (Make ONE choice and fill in a starting location, if appropriate.)

○ DFS
○ BFS
○ Prim's
○ Dijkstra's

starting from $\boxed{}$

(b) **[3 marks]** Briefly (using a few words or expressions) describe the adapted version of the algorithm you chose in part (a). Decompose your description into pre-processing, in-algorithm, and post-processsing steps. Assume that we know how the algorithms in part (a) work, and focus your description on additions and changes to them.

- pre-processing:

- in-algorithm:

- post-processing:

(c) **[3 marks]** If the number of vertices in the input graph is n, and the number of edges is m, what is the running time of your algorithm? In the three boxes below, write asymptotic running times for the pre-processing, in-algorithm, and post-processing parts of the solution. (Do not assume anything about the relationship between n and m.)

$\boxed{} + \boxed{} + \boxed{}$

(d) **[2 marks]** Do you think a graph representing the roadmap of BC is ○ sparse, ○ dense, or ○ neither? (Give a brief justification.)

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.