CS 301: Final Exam

For full credit, your answers should be clear and concise and all code should be reasonably efficient in terms of both space and time.

Problem 0 (4 points): Write your name at the top of this page.

Problem 1 (10 points): Given the following declarations state whether each statement below is syntactically legal or not.

```c
int k, m, n;
int *x, *y, *z;
```

(a) `x = new int;`
(b) `*x = *y;`
(c) `&x = k;`
(d) `delete m;`
(e) `y[8] = new int;`
(f) `m = *z;`
(g) `z[0] = k;`
(h) `n = new x;`
(i) `x = y;`
(j) `z = &y;`

Problem 2 (12 points):

(a) Show the AVL tree that results from inserting elements in the following order: 20 10 30 25 22 24.

(b) True or false: every subtree of an AVL tree is also an AVL tree. Justify your answer.

(c) True or false: If a node’s left subtree and right subtree are both AVL trees then the tree rooted at that node must also be an AVL tree. Justify your answer.

(d) True or false: An empty tree is an AVL tree. Justify your answer.
Problem 3 (12 points):

(a) Given the following hash values and the following table, determine for each of the five keys how many probes it will take to determine if that key is in the table or not. Use linear rehashing.

<table>
<thead>
<tr>
<th>Hash Key</th>
<th>Hash Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>dodge</td>
<td>1</td>
</tr>
<tr>
<td>ford</td>
<td>1</td>
</tr>
<tr>
<td>honda</td>
<td>4</td>
</tr>
<tr>
<td>toyota</td>
<td>4</td>
</tr>
<tr>
<td>vw</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>honda</td>
</tr>
<tr>
<td>1</td>
<td>ford</td>
</tr>
<tr>
<td>2</td>
<td>vw</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>toyota</td>
</tr>
</tbody>
</table>

(b) Find an ordering in which *honda*, *ford*, *vw*, and *toyota* could have been inserted in order to end up with the table shown in (a).
Problem 4 (12 points):

(a) Draw the heap that would be stored in an array as 3 5 7 6 12 14 8 13.

(b) Draw the heap from (a) after two extract-min (deque) operations have been performed.

(c) Draw the heap from (b) after adding 15 and then 1.
Problems 5 and 6 refer to the following definitions of a Queue class.

class Queue {
    public:
        typedef int ItemType;

        Queue();
        Queue(const Queue&); // copy constructor

        bool isFull() const;
        bool isEmpty() const;
        int getSize() const;

        void makeEmpty();
        void enqueue(const ItemType& toAdd);
        void dequeue(ItemType& dequeued);
        ItemType peek(); // returns next item w/o removing it

    private:
        // Use the following declarations for Problem 5

        struct Node {
            ItemType info;
            Node* next;
        };

        Node *first; // pointer to first node
        Node *tail;  // pointer to last node
        int numItems; // number of items on queue

        // Use the following declarations for Problem 6

        struct Pair {
            ItemType info;
            int count;
        };

        Pair *items; // dynamically allocated array
        int numItems; // number of items on queue
        int maxPairs; // size of array
        int numPairs; // number of used elements in array

        // NOTE: there is no first/last used for wraparound;
        // dequeue is simply going to be slow (and problem 6 easier)
};
Problem 5 (12 points): A queue of integers is said to be a **breadth-first queue** if all the elements have the same value or if all of the elements at the front of the queue are one less than all the elements at the end and there is nothing else in between (so [], [1 1], and [1 1 1 2 2] are all breadth-first queues but [1 2 2 2 3 3] and [1 1 3 2 2] are not).

(a) Write a method `int massDequeue(ItemType& dequeued)` that removes all equal values from the front of the queue, returning the number removed and copying the value to the reference parameter. The precondition for `massDequeue` is that the queue is not empty.

(b) Write a client function that takes a queue of `ints` as a parameter and returns true exactly when that queue is a breadth-first queue. The queue should not be changed. (Hint: use `massDequeue`.)
Problem 6 (14 points): To save space when the same value is repeatedly added to a queue, we can keep track of a repeat count for each item that records how many consecutive values equal to that item are on the queue. Consider such an implementation that uses a dynamically allocated array of (info, count) pairs. [] would be a queue with an empty array. [1 2] and [1 1 2 2] would both be stored using two array elements; the counts would be 1 and 1 in the first case and 2 and 3 in the second. [1 2 1] would use 3 array elements with counts all 1.

(a) Write the enqueue method for this new implementation. You may assume that you can use the == and = operators with ItemType. enqueue should resize the array if necessary.

(b) Write a member function that determines if a Queue is a breadth-first queue. You should assume that ItemType is int. The resulting function should have a better asymptotic worst-case running time than the client function you wrote in part (a).

(c) What is the asymptotic worst-case running time of your member function?
Problem 7 (12 points): Suppose there is an ADT that has two different implementations, X and Y. The ADT specifies three methods foo, bar, and baz. The asymptotic worst-case running times for each method using each implementation are given in the table below.

<table>
<thead>
<tr>
<th></th>
<th>foo</th>
<th>bar</th>
<th>baz</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(log n)</td>
</tr>
<tr>
<td>Y</td>
<td>O(log n)</td>
<td>O(log n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>

(a) Suppose algorithm A makes O(n log n) calls to foo, O(1) calls to bar and O(n) calls to baz. Which implementation of the ADT gives a better asymptotic worst-case running time for A? Explain your answer.

(b) Fill in the blank to make the following statement true: if algorithm B makes O(n^2) calls to foo, O(1) calls to bar, and ________ calls to baz then using implementation Y gives a better worst-case asymptotic running time for B.
Problem 8 (12 points):

(a) Write a function that takes a pointer to a Node in a binary tree not a binary search tree) as a parameter and returns true exactly when all nodes in the subtree rooted have keys that are less than the keys in their children (that is, the nodes obey the heap ordering property). The Node structure has three members key, left, and right. You may assume that the key values are ints.

(b) Write a function void traverseBottom(Node *tree, Queue& q, int depth) that adds to the queue, in right to left order, the depths of all nodes in the subtree rooted at tree that don’t have two children.