2017 Block 4 – Data Structures – Practice Final

The actual final should represent your individual understanding of the course material. As such, the final should be done independently and will be closed book, closed notes, closed Internet, closed phone, close friends, etc.

The practice final is shorter than the actual final is likely to be (instructor was pressed for time), but demonstrates the basic shape and structure questions on the final are likely to take.

Question 1:
8 pts - Look at the following code and consider how the memory changes as the main method executes. Draw the memory diagram for the state of the memory just before the main method
finishes execution.

ADDRESS	TYPE	NAME	VALUE
0x01	String[]	args	received from caller
0x02	Point	s	0x06
0x03	Point	e	0x08
0x04	LineSegment	l	0x0A
0x05			
0x06	int	x	0
0x07	int	y	0
0x08	int	x	5
0x09	int	y	10
0x0A	Point	a	0x02
0x0B	Point	b	0x03

```
public class Point {
    int x;
    int y;

    public Point(int a, int b) { x=a; y=b; }
}

public class LineSegment {
    Point a;
    Point b;

    public LineSegment(Point x, Point y) { a=x; b=y; }
}

public class Test {
    public static void main(String[] args) {
        Point s = new Point(0,0);
        Point e = new Point(5,10);
        LineSegment l = new LineSegment(s,e);
        // What does the memory look like here, just before
        // main returns?
        }
}
```

Question 2:
3 pts - As computer scientists, we frequently work with problems, algorithms, and programs. How are these ideas related to one another?

Problems are solved by algorithms that are implemented by a program.

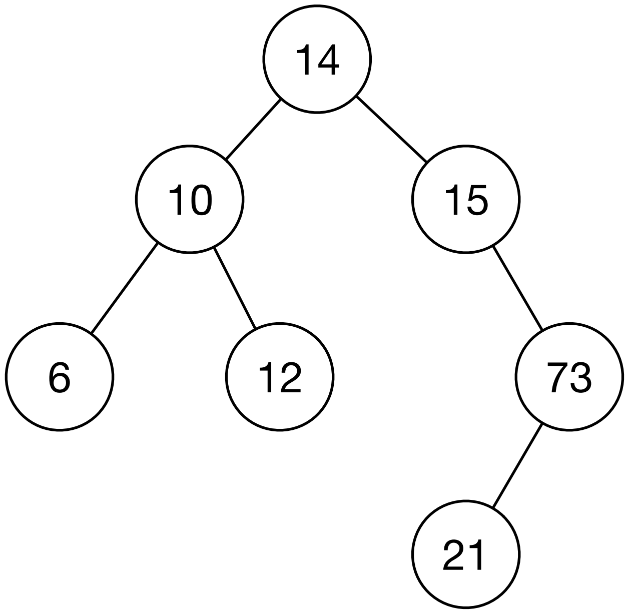
Question 3:
2 pts - If two different sorting algorithms have O(n^2) time complexity does that imply that implementations of these algorithms will take the same amount of time to sort lists of the same size? Explain your answer.

They would not take exactly the same amount of time, rather a similar amount of time. This is because an algorithm that takes (n^2 + n) amount of time would be classified as O(n^2), but wouldn't take the same amount of time to perform as an algorithm whose time complexity is (100n^2), which also can be defined as O(n^2). However, as n significantly increases (or reaches infinity), the n in (n^2 + n) and the constant in (100n^2) become irrelevant. The significant time increase is n^2 for both, which makes both of these algorithms' time complexity similar, but not identical.

Question 4:
2 pts - JAVA supports two ways to check for equality between objects, '==' and 'equals()'. Do both of these ways to check for equality produce the same result? Please explain.

The '==' and 'equals()' do not product the same result because '==' checks for address comparison while 'equals()' checks for content comparison. The 'equals()' is a method that can be overridden, while '==' is an operator.

Question 5:
Use the following tree for the sub-questions below.



1 pts - What would the output be for a pre-order traversal?

14, 10, 6, 12, 15, 73, 21

1 pts - What would the output be for an in-order traversal?

6, 10, 12, 14, 15, 21, 73

1 pts - What would the output be for a post-order traversal?

6, 12, 10, 21, 73, 15, 14

1 pts - Is the tree full or complete? Explain your answer.

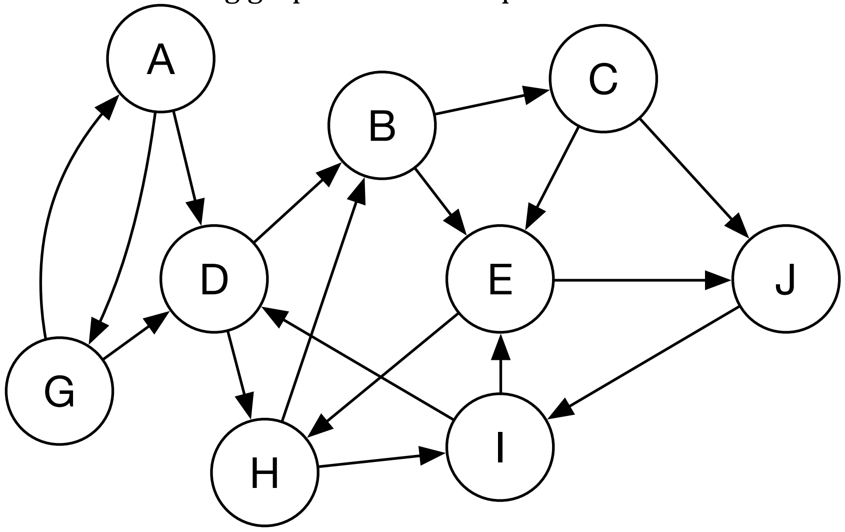
This tree is not full because not every node (except the leaves) have exactly two children. It is not complete either because not every level is filled. Node 15 is missing one child and Node 21 should be all the way to the left for it to be a complete tree.

1 pts - Does this tree look like it represents a binary search tree? Explain your answer.

Yes, this tree does represent a binary search tree because:

- every node has no more than two children
- every node is either a leaf or a root of another node
- the left side of the tree has nodes with less value than the root, and the right side has nodes with greater value than the root.

Question 6:
Use the following graph for the sub-questions below.



1 pts - Perform a depth first search from node A to E, write down the path you found.

A – D – B – C – J – I – E

1 pts - Perform a breadth first search from node G to E, write down the path you found.

Search: G – D – A – B – H – C – E
Path: G – D – B – E

1 pts – Does this graph more than one cycle? If so, write out paths for 2 cycles.

A – G – A
H – B – E – H

Question 7:
Alex, the bear, had a really hard time with lists but feels good about the queue implementation. On a new project, Alex needs list operations that work by indexes and proposes the following list implementation.

```
public class QueueList<T> {
    Queue<T> q; // a queue to hold the list
    int size;   // number of elements in the list

    public QueueList() {
        q = new Queue<T>();
        size = 0;
    }

    public T fetch(int idx) {
        T r = null;
        Queue<T> tmp = new Queue<T>();
        int i=0;
        while(i<idx) { tmp.enqueue(q.dequeue()); i++; }
        r = q.dequeue();
        tmp.enqueue(r);
        i++;
        while(i<size) { tmp.enqueue(q.dequeue()); i++; }
        q = tmp;
        return r;
    }

    public void remove(int idx) {
        Queue<T> tmp = new Queue<T>();
        int i=0;
        while(i<idx) { tmp.enqueue(q.dequeue()); i++; }
        q.dequeue();
        i++;
        while(i<size) { tmp.enqueue(q.dequeue()); i++; }
        q = tmp;
        size--;
    }

    public void append(T v) {
        q.enqueue(v);
    }

    public void insert(int idx, T v) {
        Queue<T> tmp = new Queue<T>();
        int i=0;
        while(i<idx) { tmp.enqueue(q.dequeue()); i++; }
        q.enqueue(v);
        while(i<size) { tmp.enqueue(q.dequeue()); i++; }
        q = tmp;
        size++;
    }
}
```

2 pts - Do you think Alex's list implementation looks like it will work? Argue for or against Alex's general solution.

I think Alex’s list implementation would work for the most part. The general logic of it seems reasonable. It is not the most efficient code, but it works.

4 pts - Are there defects in Alex's code? Edge cases that need to be handled or potential off by one problems? If so, high light where the problem might be and explain what might go wrong.

The append method does not increase the size variable, but it does make the list bigger. This could lead to many index out of bounds exceptions in the fetch, remove, and insert methods. Also, the insert method does the assignation backwards. I does not copy the temp into q, but copies the q into temp.

1 pts – What is the time complexity for fetch? Is it a tight bound? Justify your answer.

Time complexity would be O(n) (linear) because every time the fetch method is called, the queue gets dequeued completely and copied into a new queue. It is a tight bound because even in the best case, this fetch method creates a new copy of the queue.

1 pts - What is the time complexity for remove? Is it a tight bound? Justify your answer.

Time complexity for the remove method is also linear and also has a tight bound for the same reasons as the fetch method. Both best and worst case scenarios create a whole new copy of the queue (in this case, one index smaller).

1 pts - What is the time complexity for insert? Is it a tight bound? Justify your answer.

Insert also has a linear tight complexity because similarly to the other methods, it creates a copy of the original queue in all cases.

1 pts – Is there any setting in which Alex's list implementation would be preferable to a linked list or array list? Please explain.

No, Alex's implementation is never more efficient than an array list or a linked list. Linked lists and array lists have methods whose time complexity is constant time, which makes them more efficient than Alex's methods which are all linear.