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, closed 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.

```
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 the most general thing that computer scientists deal with. They are things that can be expressed in language like "sort a list of 100 elements". The job of the computer scientist is to translate these problems

into a format that a computer can understand. These translated problems are called "programs". An "algorithm" is a procedure employed by a program to accomplish general cases of a specific task efficiently. An example, yet again, would be a sorting algorithm like an insertion sort. The algorithm is just a part of the program that implements it, but it is likely the most important part.

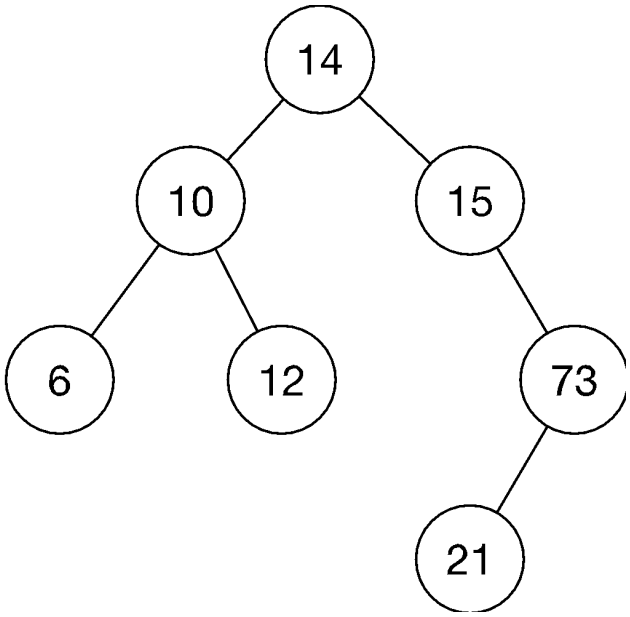
Question 3:
2 pts - If two different sorting algorithms have O(n^2) time complexity does that imply that implementations of these algorithms will take the same amount of time to sort lists of the same size? Explain your answer.

No, since the operations that each algorithm performs don't necessarily have to take the same amount of time. For example, the iterative implementation of a given sort usually takes less time that the recursive one even though they are the same sorts.

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.

No. '==' usually checks if the two objects are exactly the same object (that is, they occupy the same location in memory). Checking if two integers equal each other with '==' is usually safe, but checking two strings will often not return the result you were expecting. "Hello!" == "Hello!" returns false, but "Hello!".equals("Hello!") returns true.

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, 14, 15, 73, 21

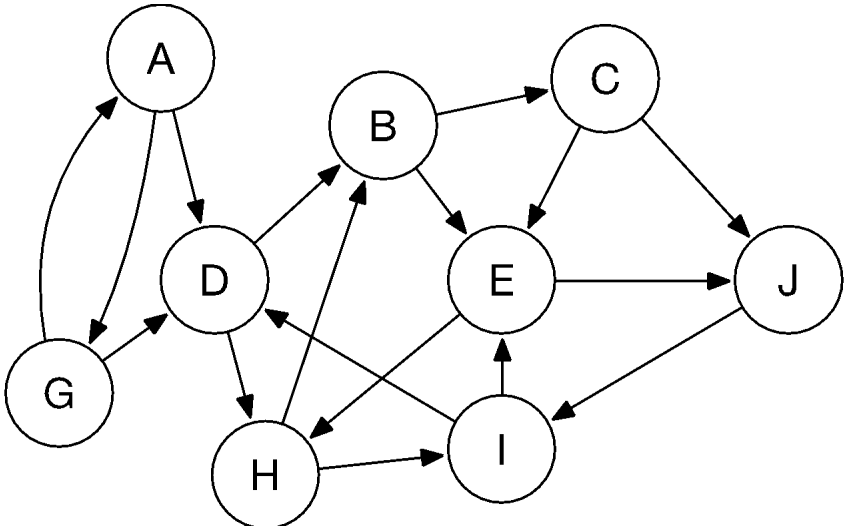
1 pts - Is the tree full or complete? Explain your answer.

No. The second-to-last level is not filled, so it cannot be complete, and a tree that is not complete definitely cannot be full.

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

It does. Each node has either two, one, or zero branches, and the left and right branches are always less and greater than the root node, respectively.

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.

I chose to always take the highest path that I haven't already searched. This lead me to try the A>D>B>C>J>I>E path first. Since that path got me to E, I returned it.

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

I started at A and searched both G and D. From G, I tried the only possible node, D, and from D I tried B and H. After trying C and E from B, I found a successful path and returned A>D>B>E.

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

It does. H>I>E>H is one, as is E>J>I>E

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.

All the methods in Alex's implementation do what they should assuming that enqueue() and dequeue() are implemented correctly.

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.

One potential problem is that negative indexes will cause all kinds of issues (like fetching the first element instead of returning an error and causing an

overflow exception when the method tries to iterate from idx to size thinking idx is 0).

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

Fetch is O(n) with a tight bound because it always performs one operation for each element in the list.

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

Remove is also O(n) with a tight bound since it performs one operation for each element in the list.

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

Insert is O(n) with a tight bound like fetch and remove since it iterates over the list and performs a single operation for each element.

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

Depending on how it is implemented, a Queue is essentially just a list where not every element is always accessible. For this reason, Alex's implementation shouldn't be more efficient than an a linked list or arraylist in terms of time complexity, but there are situations where it is more intuitive or user-friendly.