Tian Lee

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.

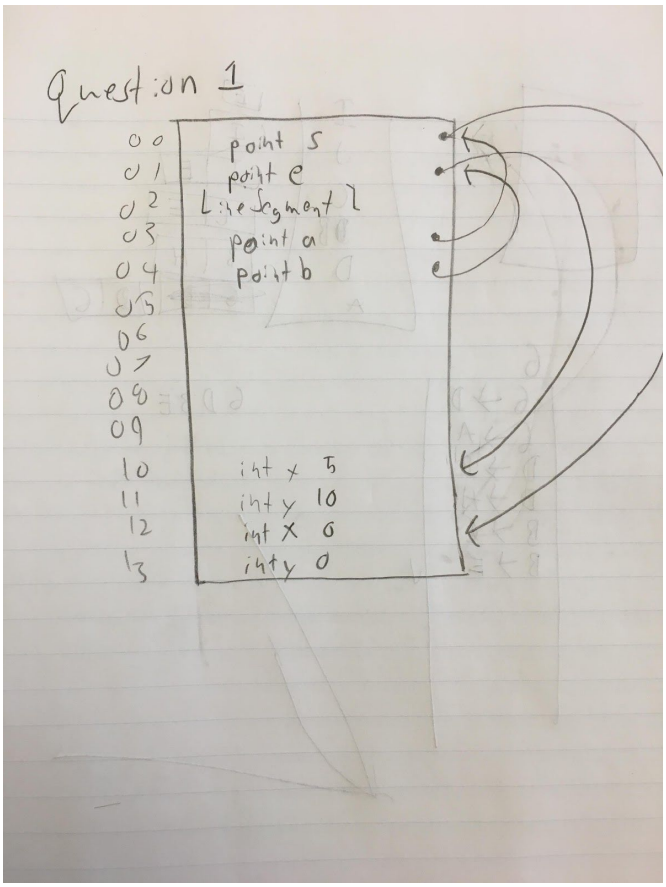
```
public class Point {
   int x;
   int y;

   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?

Typically these things are related to each other because we have a problem that can be solved with a series of algorithms. These algorithms work together and form programs that solve the provided problem.

**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.

The fact both algorithms have an upper bound of n squared does not mean that they will both finish within the same time. This is because an upper bound is an estimate of what the worst case scenario is for time complexity. So in reality these programs will probably have similar time complexities but definitely not the same time complexities.

**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.

These two operation do not yield the same result because “==” will check to see if both the inputs are pointing to the same location in memory while the “.equals()” method will compare the values of those two specific objects. In other words the “==” will yield a false if the values of the two objects are the same but their locations in memory are different.

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

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 neither full nor complete because 15 and 73 one have one child node and the non leaf levels aren’t full.
1 pts - Does this tree look like it represents a binary search tree? Explain your answer.
It may not be representative of the ideal binary search tree but It does obey the requirements for a binary search tree such as there only being two nodes allowed as children max and having each child to the left be less than the parent and every child to the right being more.

**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
1 pts - Perform a breadth first search from node G to E, write down the path you found.
G,D,B,E
1 pts – Does this graph more than one cycle? If so, write out paths for 2 cycles.
1. A,G,A
2.H,B,E,H
3.D,B,C,J,I,D

**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.
In general I think that Alex's solution is ok. That is to say you can create a functional list using temporary Queues although his insert isn't operating correctly and the append forgets to update the size.

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.
All of his code appears functional except for the insert method which, after reaching the appropriate index in the queue, enques the desired insert value into the "q" queue instead of the temporary array which means that the value is appended instead of being inserted. Additionally it doesn't update the size until the end which makes the second while loop not even append the value from the original queue. Additionally append does not update the size which would create problems in other functions after using append.

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

In this function fetch has a linear time complexity with respect to the length of the list because fetch operates by going through the whole queue saving the desired element and then copying the same queue back into the temp and reassigning q to temp. the amount of time it takes to transfer all of the elements from q to temp is dependent on the length of q. This also results in it being tight bound.

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

remove has a tight bound of linear time complexity with respect to the length of q because it transfers all of the elements from q to temp except for the desired removal index. this yields a tight bound of linear time with respect to the length of q.

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

Insert, if it worked, would yield a linear time complexity with respect to the length of q because it transfers all of the elements from q to temp with the addition of the insertion value.

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 because Alex's implementation has mostly the same time complexity for all other methods in other implementation except for fetch in an arraylist which would be faster because all it has to do is find the desired index which is not dependent on the size of the array.