Ethan Holland

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?
   }
}
```

0X00	argv
0X01	s
0X02	e
0X03	l
0X04	
0X05	X(s) 0
0X06	Y(s) 0
0X07	X(e) 5

0X08	Y(e) 10
0X09	L (Point x) s
0X0A	L(Point y) e

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

The problems that computer scientist face are essentially how to make computers, with their few, basic operations, solve complex problems. The algorithms that computer scientists write are the step by step processes that they write to solve these problems. These algorithms are implemented in a computer program which, when run, should solve the problems presented. The hardest part of computer science is writing programs that properly implement these algorithms.

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.

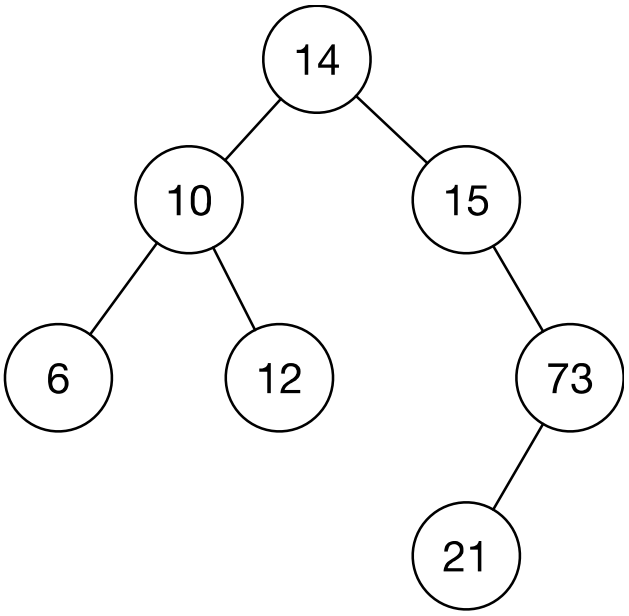
Time complexities such as O(n^2) only represent the shape of the time complexity. They do not represent the exact time complexities of different programs. Programs with O(n^2) time could be described by the formula n^2 or the formula 100n^2 which would have drastically different run times.

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 == method compares the location of objects while equals() compares whether or not the objects have equivalent values. If two object have the same values but are different objects then these methods will produce different results. Equals will return true while == will not.

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

Ethan Holland



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

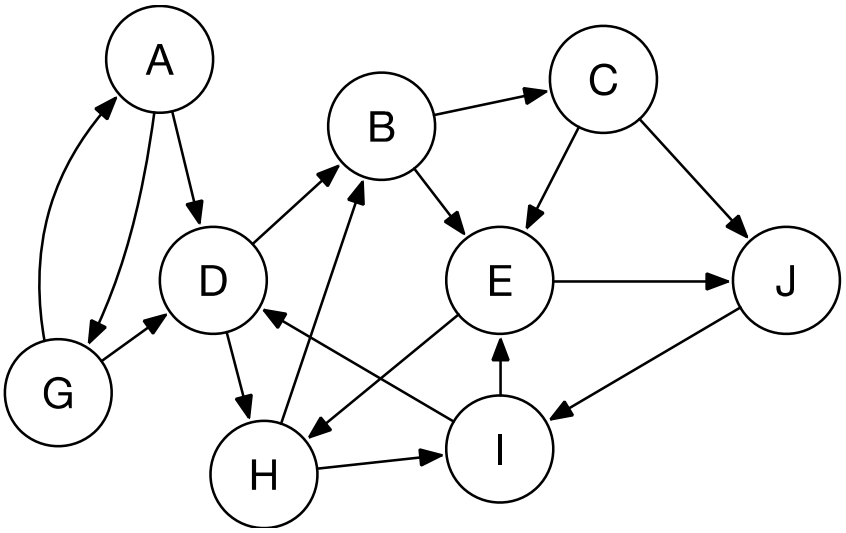
1 pts - What would the output be for a post-order traversal?
6, 12, 10, 73, 21, 15, 14

1 pts - Is the tree full or complete? Explain your answer.
The tree is not full since nodes 15 and 73 each have one child.
The tree is not complete because node 15 has a right child but no left child. Also since nodes 6 and 12 do not have any children and 73 does it is not complete.

1 pts - Does this tree look like it represents a binary search tree? Explain your answer.
This tree is a binary search tree since each node has at most two children and at any given node, all nodes to the left have smaller values and all nodes to the right have larger values.

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

Ethan Holland



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

Path A>D>B>C>E.
Rule of choice is earlier letter in alphabet.
1 pts - Perform a breadth first search from node G to E, write down the path you found.
G>A
G>D
D>B
D>H
B>C
B>E
H>I
C>J
E.

Path A>D>B>E

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

It has more than one cycle.
Cycle 1: A>G>A
Cycle 2: D>H>I>D

Question 7:

Ethan Holland

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);                                          code to  get to index
      i++;
      while(i<size) { tmp.enqueue(q.dequeue()); i++; }
      q = tmp;                                                 code to refill q and return r.
      return r;
   }

   public void remove(int idx) {
      Queue<T> tmp = new Queue<T>();
      int i=0;
      while(i<idx) { tmp.enqueue(q.dequeue()); i++; }          code to reach index
      q.dequeue();
      i++;
      while(i<size) { tmp.enqueue(q.dequeue()); i++; }         skips idx
      q = tmp;                                                 replace q
      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++;
   }
}
```

Ethan Holland

2 pts - Do you think Alex's list implementation looks like it will work? Argue for or against Alex's general solution.
Alex's general solution looks like it will work. The methods require the dequeuing of q into a temp q which appears to properly preserve the elements and order of the queue. All the basics of a list are covered by the Alex's queue list implementation.

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 update the size of the queue. If only the append method is used to enqueue values then none of the other methods will properly finish requeuing after reaching the specified index. All values after idx will not be enqueued into the temp queue and when the q=temp line executes all values after idx will be lost. If the append method is used n times to enqueue and insert is used m time to enqueue then n number of values would be lost when fetch, remove or insert is called.

1 pts - What is the time complexity for fetch? Is it a tight bound? Justify your answer.
The time complexity has a linear tight bound. When an index is fetched, the temp queue is enqueued until the index is reached. The value is returned and the rest of the temp queue is enqueued with the rest of the main queue. The operation requires n number of deques and enqueues, where n is the size of the queue, to occur.

1 pts - What is the time complexity for remove? Is it a tight bound? Justify your answer.
The time complexity for remove has a linear tight bound. Similar to the fetch command, remove requires idx-1 enqueues and deques to occur to reach the index, and requires a total of n-1 enqueues and deques to refill q.

1 pts - What is the time complexity for insert? Is it a tight bound? Justify your answer.
The time complexity has a linear tight bound. Similar to the fetch and remove command, insert requires idx-1 enqueues and deques to occur to reach the index where it then enqueues the input value and requires a total of n+1 enqueues and deques to refill q.

1 pts – Is there any setting in which Alex's list implementation would be preferable to a linked list or array list? Please explain.
There is not a setting where the queue list implementation is preferable. Since queues are array backed Alex's queue list has a maximum size. The queue list does not have constant fetch time like an array backed list since the first value is the only one that can be directly accessed.