Points off	1	2	3	4	5	6	Total off	Net Score

CS 314 – Final Exam – Spring 2016

Your Name_____

Your UTEID _____

Instructions:

1. There are **6** questions on this test. 100 points available. Scores will be scaled to 300 points.
2. You have 3 hours to complete the test.
3. Place you final answers on this test. Not on scratch paper. Answer in pencil.
4. You may not use a calculator or any other electronic devices while taking the test.
5. When answering coding questions, ensure you follow the restrictions of the question.
6. Do not write code to check the preconditions.
7. On coding questions you may implement your own helper methods.
8. On coding questions make your solutions as efficient as possible given the restrictions of the question.
9. Test proctors will not answer any questions regarding the content of the exam. If you think a question is ambiguous or has an error, state your assumptions and answer based on those assumptions.
10. When you complete the test show the proctor your UTID, give them the test and all the scratch paper, used or not, and leave the room quietly.

1. (1 point each, 25 points total) Short answer. Place your answer on the line next to or under the question. Assume all necessary imports have been made.

   a. If a question contains a syntax error or other compile error, answer "Compile error".
   b. If a question would result in a runtime error or exception answer "Runtime error".
   c. If a question results in an infinite loop answer "Infinite loop".
   d. Recall when asked for Big O your answer shall be the most restrictive correct Big O function. For example Selection Sort has an average case Big O of $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of $O(N^3)$ or $O(N^4)$. I want the most restrictive, correct Big O function. (Closest without going under.)

A. When n = 200 it takes the following method 1 second to complete. What is the expected time for the method to complete when n = 400?

```
public int a(int n) {
     int sum = 0;
     int ns = n * n;
     for (int i = 0; i < n; i++) {
          for (int j = 0; j < i * ns; j++) {
               sum++;
          }
     }
     return sum;
}
```

_____

B. What is returned by the method call `b(12)`? _____

```
public int b(int x) {
     if (x <= 2)
          return x * 2;
     else
          return b(x - 3) + 2;
}
```

C. What is returned by the method call `c(3, -2)`? _____

```
public int c(int x, int y) {
     if (x <= 2)
          return 1;
     else {
          int newY = y + 1;
          return c(x - y - 1, newY) + c(x - y - 2, newY);
     }
}
```

D. What is output by the following code? The code uses the queue class we developed in lecture.

_____

```
int[] data = {3, -1, 2, -2, 2, 4, 1, 4};
Queue<Integer> q = new Queue<Integer>();
for (int x : data) {
     for (int i = 0; i < x; i++) {
          q.enqueue(x);
     }
}

int result = 0;
for (int i = 0; i < 9; i++) {
     result += q.dequeue();
}

System.out.println( result + " " + q.isEmpty());
```

E. What is the result of the following postfix expression? (single integer for answer) _____

```
40 10 / 5 3 - 4 * 3 + *
```

F. A method uses the mergesort to sort arrays of `int`s into ascending order. It takes the method 0.2 seconds to sort an array of 1,000,000 distinct `int`s that are initially in descending order. How long do you expect the method to take when sorting an array of 2,000,000 distinct `int`s that are initially in descending order?

_____

G. Consider the following implementation of a `PriorityQueue`. What is the average case order (Big O) of the enqueue method if the `PriorityQueue` already contains N elements?

_____

```
public class PriorityQueue<E extends Comparable<E>> {

     private ArrayList<E> con;

     public PriorityQueue() {
          con = new ArrayList<E>();
     }

     // Determines index to insert val into con using the binary
     // search algorithm. Implementation not shown.
     private int insertionIndex(E val)

     public void enqueue(E val) {
          int index = insertionIndex(val);
          con.add(index, val);
     }
}
```

H. This question uses the same `PriorityQueue` class from question 1.G. What is the order (Big O) of the `dequeue` method if the `PriorityQueue` contains N elements?

_____

```
// part of the PriorityQueue from question 1.G
public E dequeue() {
     E result = con.remove(0);
     return result;
}
```

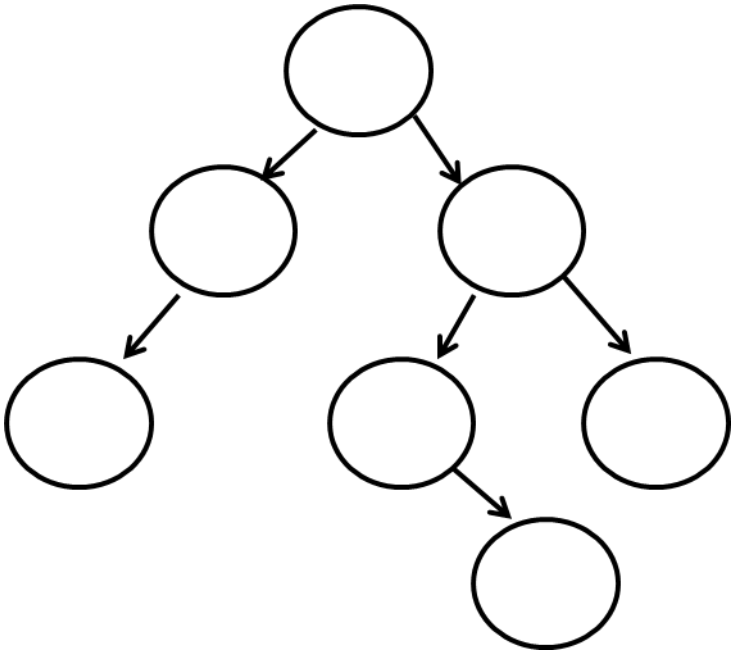
I. What changes could be made to the `PriorityQueue` class to improve the efficiency of the `dequeue` method from question 1.H without changing any of the code in the `dequeue` method itself? **Limit yourself to a single sentence.**

_____

J. What changes could be made to improve the efficiency of the `dequeue` method from question 1.H by changing the code of the `dequeue` itself without changing the actual data type of the internal storage container of the `PriorityQueue` itself? **Limit yourself to a single sentence.**

_____

K. Fill in each node in the binary tree shown below with an integer value so that the tree is a binary search tree.



L. The following values are added one at a time to an initially empty binary search tree using the traditional, naïve insertion algorithm. What is the result of a pre order traversal of the tree?

```
13, 0, 5, -5, 0, 13, 12, 11
```

_____

M. The following values are added one at a time to an initially empty binary search tree using the traditional, naïve insertion algorithm. What is the height of the resulting tree?

```
15, 12, 17, 6, 12, 9, 0, 7, 7
```

_____

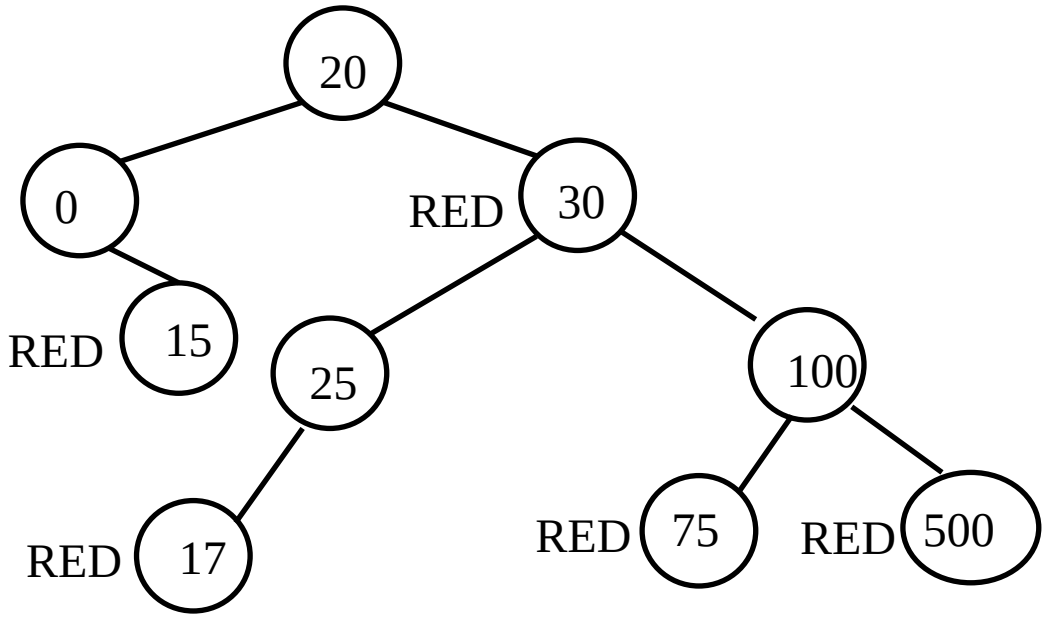
N. The following method takes 15 seconds when `data.size()` = 10,000.
What is the expected time for the method to complete when `data.size()` = 20,000?
N = `data.size()`

_____

```
public String n(HashSet<Integer> data) {
     String result = "";
     for (int i : data) {
          if (i % 2 == 0) {
               result += i + " ";
          }
     }
     return result;
}
```

O. Is the following tree a Red-Black tree? Nodes not labeled RED are BLACK.

_____



P. Given the following codes for the characters shown, draw the resulting Huffman Tree.

```
'a': 1101
'e': 0
'l': 1100
's': 10
't': 111
```

Q. The following method takes 10 second to complete when the array contains 1,000,000 distinct elements in random order. What is the expected time for the method to complete when the array contains 4,000,000 distinct elements in random order? The `TreeSet` class is the `java.util.TreeSet` class.

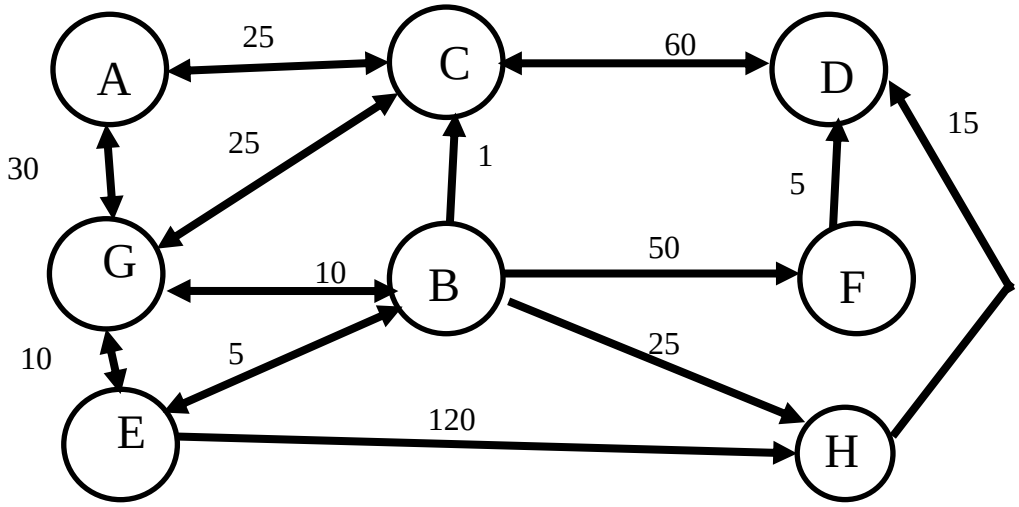
```
public TreeSet<Integer> q(int[] data) {
     Arrays.sort(data);
     TreeSet<Integer> result = new TreeSet<Integer>();
     for (int x : data) {
          result.add(x);
     }
     return result;
}
```

R. Why don't implementations of Graphs typically use an adjacency matrix to store the edges of the Graph?

S. What do heap data structures typically use as their internal storage containers?

_____

T. Consider the following weighted, directed Graph:



What is the cost of the lowest cost path **from** vertex A **to** vertex D? _____

U. Is the graph shown in 1.T a directed acyclical graph? _____

V. The following method takes 3 seconds to complete when `set` and `data` both contain 500,000 distinct values and roughly half of the values in `data` are in `set`. What is the expected time for the method to complete when `set` and `data` both contain 2,000,000 distinct values and roughly half of the values in `data` are in `set`?

_____

```
public int remove(HashSet<Integer> set, int[] data) {
     int result = 0;
     for (int x : data) {
          if (set.remove(x)) {
               result++;
          }
     }
     return result;
}
```

W. The following method takes 10 seconds to complete when `n` = 40. What is the expected time for the method to complete when n = 50?

```
public int fib(int n) {
     if (n == 1 || n == 2)
          return 1;
     else
          return fib(n - 2) + fib(n - 1);
}
```

X. The following values are added in the order shown to a min heap using the algorithm shown in class. Draw the resulting heap.

`4, 10, 8, 10, 10, 10, 7`

Y. What is the average case order (Big O) of the following code assuming `list` contains N distinct elements in random order?

```
public List<Integer> y(LinkedList<Integer> list) {
     LinkedList<Integer> result = new LinkedList<Integer>();
     for (int i = 0; i < list.size(); i++) {
          int value = list.get(i);
          boolean present = false;
          int j = 0;
          while(!present && j < result.size()) {
               present = result.get(j) == value;
               j++;
          }
          if (!present) {
               result.add(0, value); // position, value
          }
     }
     return result;
}
```

2. Stacks - 15 points. Write a method that determines if the values in a stack are in ***descending*** order from top to bottom.

Consider these examples with stacks of integers.

```
empty stack -> returns true

top: 5 -> returns true
     5

top: 5 -> returns true
     5
     3
     -1

top: 5 -> returns false
     3
     1
     4
     0

top: 12 -> returns true

top: 5 -> returns false
     10

top: 5 -> returns true
     5
     5

top: 10 -> returns false
     20
     30
     25
     50
```

The `Stack` class has the following methods:

`boolean isEmpty(), void push(E value), E top(), E pop()` and a default constructor that creates an empty stack.

You may also use the `compareTo` method from the `Comparable` interface.

You may use create and use one auxiliary `Stack`.

The `Stack` given as a parameter must be restored to its original state by the completion of the method.

Complete the method on the next page.

The method is **not** an instance method of the `Stack` class.

```
/* pre: st != null
   post: return true if the elements in st are in descending order
     according to the natural ordering of the data type. st is
     restored to its original state. */
public static <E extends Comparable<E>> boolean isDescending(Stack st)
```

3. Linked Lists (15 points) - Complete the `remove(int value)` instance method for a `SortedIntList` class. The method removes all values from the `SortedIntList` equal to the given parameter and returns the number of elements removed. The `SortedIntList` class uses a linked structure of nodes as its internal storage container. The `SortedIntList` class stores `int`s in sorted, ascending order.

- You may not use any other methods from the `SortedIntList` class unless you implement them yourself as a part of your solution.
- The `SortedIntList` class uses singly linked nodes.
- The list has a reference to the first node in the linked structure of nodes.
- When the list is empty, `first` is set to `null`.
- If the list is not empty the last node in the list has its next reference set to `null`.
- You may use the nested `Node` class.
- **You may not use any other Java classes or native arrays.**

```
public class SortedIntList {

     // Refers to first node in the chain of nodes.
     private IntNode first;

     // No other instance variables

     // The nested IntNode class.
     private static class IntNode {
          private int data;
          private IntNode next;
     }

}
```

Examples. of `int remove(int value)`

```
[].remove(5) returns 0, resulting list -> []

[5, 5, 5].remove(5) returns 3, resulting list -> []

[1, 1, 7, 7].remove(5) returns 0, resulting list -> [1, 1, 7, 7]

[1, 1, 4, 5, 5].remove(1) returns 2, resulting list -> [4, 5, 5]

[-5, -5, -3, 0, 0].remove(1) returns 0,
                              resulting list -> [-5, -5, -3, 0, 0]

[5, 5, 6, 11].remove(1) returns 0, resulting list -> [5, 5, 6, 11]
```

Complete the int `remove(int value)` instance method for the `SortedIntList` class on the next page.

```
/* pre:  none
   post: Remove all occurrences of value from this list and return
          the number of values removed by this method.
*/
public int remove(int value) {
```

4. Binary Trees - 15 points. Complete the `numNodesLessThanSumOfAncestors` accessor method for a binary tree of `int`s class. The binary tree class for this question stores `int`s, but it is **not** a binary search tree.

The method returns the number of nodes in the tree that contain a value that is less than the sum of the values in that node's ancestor nodes. Recall ancestor nodes are all the nodes in the path from the root down to a node. A node is not an ancestor of itself.

You may use the nested `BNode` class, but no other Java classes or methods. Not even arrays.

Consider the following trees and the expected results. The nodes that meet the criteria are bolded.

```
empty tree -> returns 0

5 -> returns 0      -5 -> returns 1 (sum of ancestors is 0)

   5 -> returns 1                        5 -> returns 1
  / \                                   /
13   4                                 2



   5 -> returns 0                        5 -> returns 2
  / \                                   / \
 12  5                                -5   4


                          5             -> returns 4
                        /   \           -5 < 5
                      -5     15         -1 < 0 (5 + -5)
                     /  \   /  \        -6 < -2 (5 + -5 + -1 + -1)
                    3   -1 25   30     840 < 850 (5 + 15 + 30 +
                       /          \                100 + 200 + 500)
                     -1            100
                    /             /
                   -6           200
                                  \
                                  500
                                    \
                                    840
```

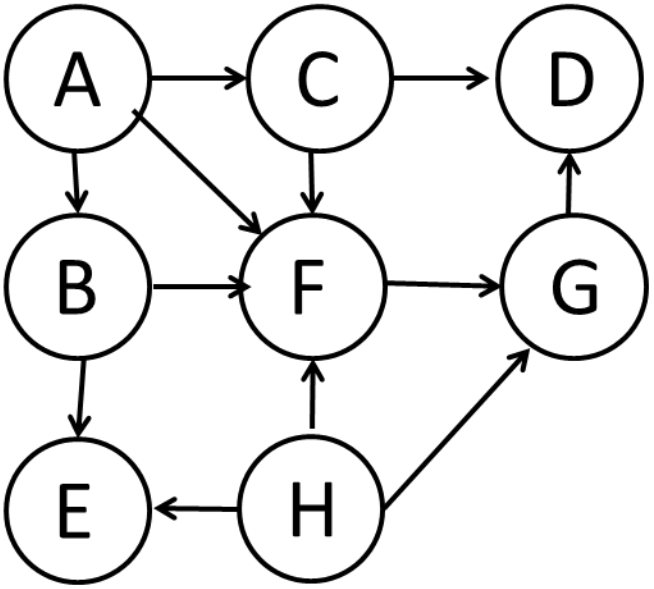
```
public class BinaryTree {
     private BNode root; // root == null iff tree is empty

     private static class BNode {
          private int value;
          private BNode left;
          private BNode right;
     }
}
```

Complete the following instance method for the **`BinaryTree`** class. You may not use any other methods from the **`BinaryTree`** class unless you implement them as part of your answer.

```
/* pre: none
   post: Returns the number of nodes in this tree that contain a value
           that is less than the sum of the values in the node's
           ancestor nodes. */
public int numNodesLessThanSumOfAncestors() {
```

5. Maps and Graphs - 15 points. In a directed graph the ***indegree*** of a vertex is the number of edges that lead into that vertex. Consider the following graph:



The indegree of each vertex in the graph to the left is:

```
A: 0
B: 1
C: 1
D: 2
E: 2
F: 4
G: 2
H: 0
```

Write an instance method for the `Graph` class from assignment 12 that updates the `scratch` instance variable for each `Vertex` object in the `Graph` so that the `scratch` variable correctly stores the indegree of the `Vertex`. The method also returns the largest indegree in the `Graph`. In the example above the method would return 4.

```
public class Graph {
    // The vertices in the graph.
    private Map<String, Vertex> vertices;

    private int updateIndegree() // complete this method

    private static class Vertex {
        private String name;
        private List<Edge> adjacent;
        private int scratch; // use to store indegree of Vertex
    }

    private static class Edge {
        private Vertex dest;
        private double cost;
    }
}
```

You may use the `Map, Vertex, Edge, List, Iterator`, and `Set` classes.

You may not use any other methods from the Graph class unless you implement them as part of your solution. Do not use recursion in your solution.

There is no precondition on what the `scratch` variable in each `Vertex` object stores before this method is called. In other words, the `scratch` variables are not necessarily store 0 initially.

```
/* pre: none
   post: Each Vertex's scratch variable stores the indegree
     of the Vertex. Returns the largest indegree in the Graph. */
private int updateIndegree() {
```

6. Graphs - 15 points. Write a method that prints out a ***topological sort*** of a graph.

A topological sort is a linear ordering (think list) of vertices in a graph such that if an edge exists between two vertices u and v, u appears before v in the linear ordering. u does not have to be immediately before v, just somewhere before it in the linear ordering.

A topological sort is only possible if the graph is a directed acyclical graph.

Consider the example graph from question 5. It is a directed acyclical graph.

One topological sort of its vertices is: A H C B F E G D.

There are typically, but not always, multiple topological sorts for a given graph.

Another topological sort for the graph from question 5 is: H A B C E F G D.

There are many algorithms for producing a topological sort of a graph.

Here is the pseudocode for one of them:

```
add all the vertices with indegree of 0 to a queue
while the queue isn't empty
     dequeue a vertex
     add this vertex to the result
     for all of this vertex's edge, reduce the indegree of the
          destination vertex by one
          if the indegree of the destination is 0, enqueue it
return result
```

Implement the above pseudocode in the method on the next page.

You may use the `Map`, `Vertex`, `Edge`, `Queue`, `ArrayList`, `Iterator`, and `Set` classes.

Assume the `Queue` class only has the `enqueue`, `dequeue`, `isEmpty`, and `front` methods.

This question uses the same `Graph`, `Vertex`, and `Edge` classes as question 5.

Assume the `updateIndegree` method works.

You may not use any other methods from the Graph class, other than `updateIndegree`, unless you implement them as part of your solution.

Do not use recursion in your solution.

```
/* pre: This Graph is a directed acyclical graph.
   post: return a list with a topological
         sort of the vertices of this Graph. */
public ArrayList<String> getToposort() {
     this.updateIndegree();
```