Points off	1	2A	2B	2C	3	4	5	Total off	Net Score

CS 314 – Final – Fall 2011

Your Name_____

Your UTEID _____

Instructions:

1. There are **5** questions on this test.
2. You have 3 hours to complete the test.
3. You may not use a calculator or any other electronic devices while taking the test.
4. When writing a method assume the preconditions of the method are met.
5. When writing a method you may add helper methods if you wish.
6. When answering coding questions ensure you follow the restrictions of the question.
7. When you complete the test show the proctor your UTID. Give them the test and any scratch paper. Please leave the room quietly.

1. (2 points each, 40 points total) Short answer. Place you answers on the attached answer sheets.
   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 should 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. What is returned by the method call `a(4)`?

```
public int a(int x) {
     if(x == -2)
          return x;
     return x + a(x - 1) + x;
}
```

B. What is returned by the method call `b(0, new int[]{4, 2, 3, 6, 8, 3, 4, 0, 6})`?

```
public int b(int i, int[] data) {
     if(i >= data.length - 1)
          return -2;
     else if(data[i] % 2 == 0)
          return 2 + b(i + 2, data);
     else
          return 1 + b(i + 1, data);
}
```

C. The following values are inserted one at a time into a binary search tree that uses the traditional, naïve insertion algorithm. What is the result of a post order traversal of the resulting tree?

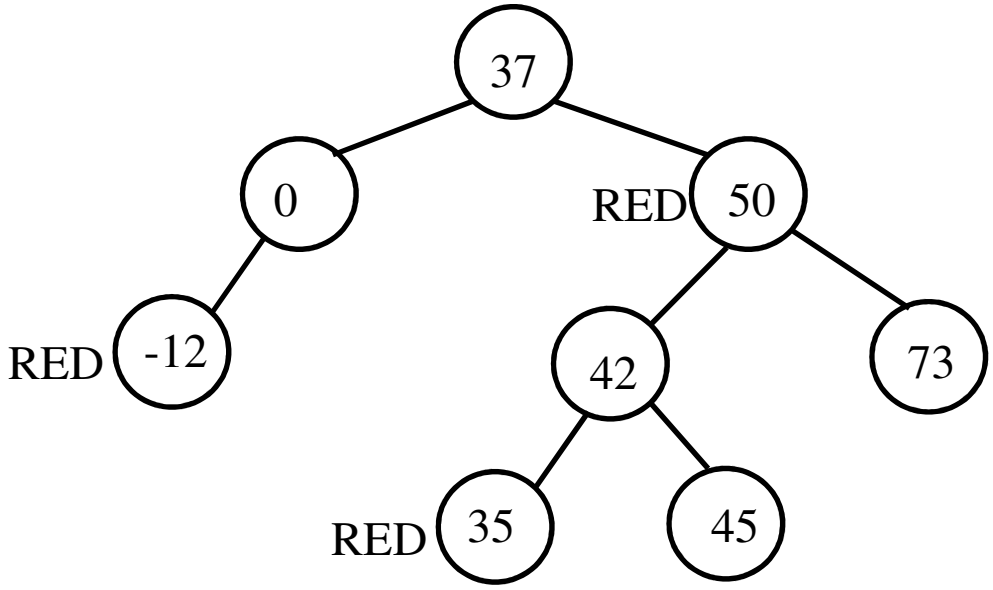
```
4, 3, 6, 17, 15
```

D. On the answer sheet fill in values for the 5 nodes so the resulting tree is a binary search tree.

E. The values 1, 2, 3, 4, …, N are inserted one at a time into an initially empty binary search tree that uses the traditional, naïve insertion algorithm. What is the order (Big O) to do all the insertions?

F. The values 1, 2, 3, 4, …, N are inserted one at a time into an initially empty Red-Black tree. What is the order (Big O) to do all the insertions?

G. Consider the following tree. The tree is does not meet the requirements of a Red-Black Tree. State why not. Be specific. In the drawing below, all nodes not labeled RED are colored BLACK.



H. Given the `sort_H(int data[])` method consider the following timing data:

- 1 second to sort an array with 500,000 distinct elements in random order.
- 9.3 seconds to sort an array with 4,000,000 distinct elements in random order

- 9.3 seconds to sort an array with 4,000,000 distinct elements in descending order
- 40.6 seconds to sort an array with 16,000,000 distinct elements in descending order

Based on the timing data, what sorting algorithm does method `sort_H` most likely use? Consider only the sorts we discussed in class.

I. Consider the following method header:

```
/* pre: targets != null, source != null
   post: return the total number of times the elements in targets
     are present in source. For example, if targets = [1, 3, 1]
     and source = [2, 3, 1, 1, -2, 5, 13] then the method would
     return 5. (First 1 in targets present twice in source, 3 in
     targets present once is source, and second 1 in targets
     present twice in source.) */
public int search(int[] targets, int[] source) {
```

Method `search` uses the linear search algorithm. When `targets.length = 300,000` and `source.length = 1,000,000`, method `search` takes 5 seconds to complete. What is the expected time for method `search` to complete when `targets.length = 600,000` and `source.length = 4,000,000`.

J. Briefly explain how collisions are resolved in a hash table that uses closed address hashing.

K. In a hash table that uses open address hashing what is the average case order (Big O) of the add method given the hash table already contains N values and has a load factor of 0.5? Explain what happens to the order of the add method if the load factor of the hash table is close to 1.0.

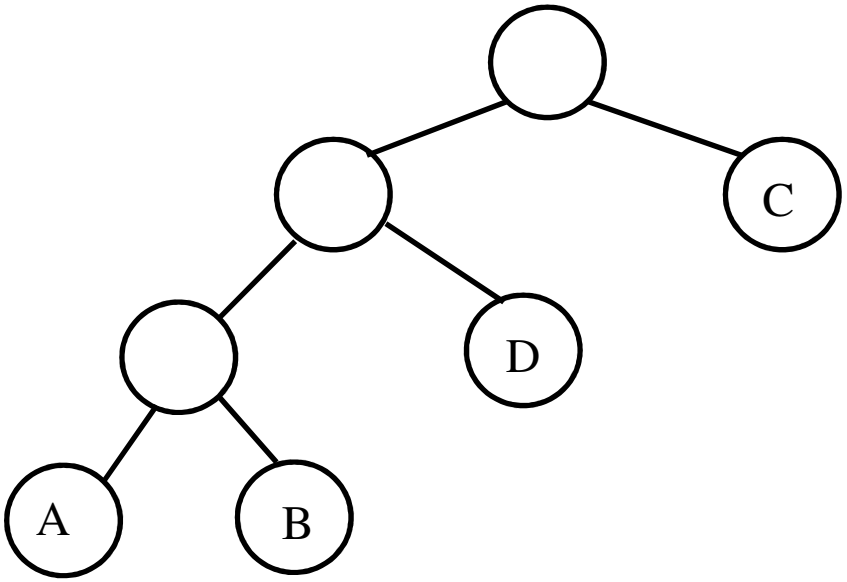
L. In Java, what types of objects may be added to a hash table? Briefly explain why.

M. What is output by the following code? The `add` method is equivalent to enqueue and the `remove` method is equivalent to dequeue.

```
String[] data = {"C", "Z", "A", "Z", "B", "Z"};
PriorityQueue<String> pq = new PriorityQueue<String>();
for(String s : data)
     pq.add(s);

while(!pq.isEmpty())
     System.out.print(pq.remove());
```

N. Consider the following Huffman Code tree.

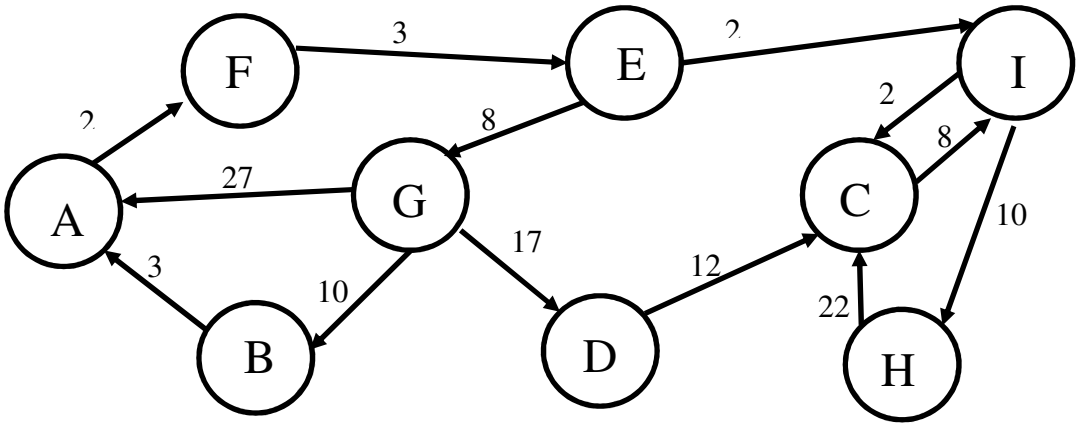


Now consider the following message encoded with the above tree. (Spaces shown to make it more readable.) What does the message decode to?

```
0 0 0 1 0 1 1 0 1 0 0 0 0 1
```

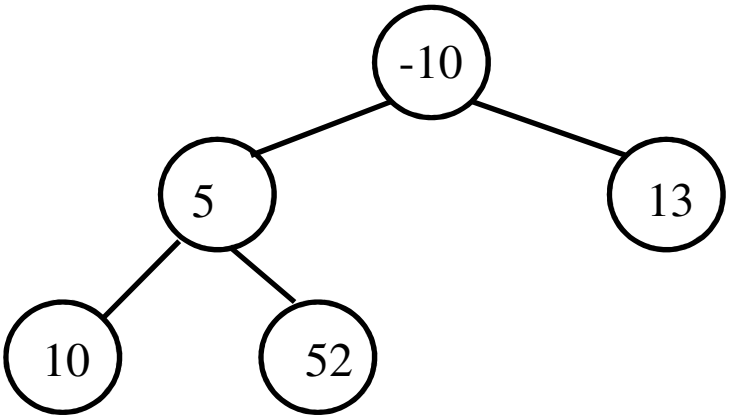
O. Briefly describe two reasons a file may not be smaller when encoded and "compressed" using Huffman encoding.

P. Consider the following graph. What is the cost of the shortest path from vertex G to vertex H?

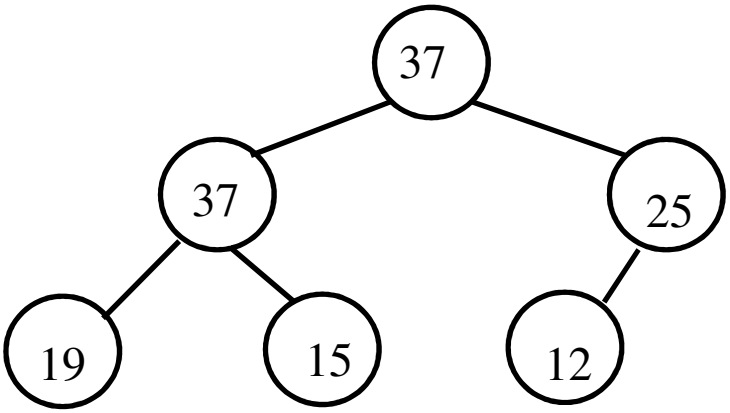


Q. A directed, weighted graph is implemented with an adjacency matrix to store the cost of edges between vertices. The matrix is N by N where N equals the number of vertices in the graph. (There is no extra capacity in the matrix.)What is the order (Big O) to add an edge to the graph between two vertices that were not previously in the Graph?

R. Consider the following min heap. If the value `15` is added to the heap what is the resulting heap? Use the add method shown in class.



S. Consider the following max heap. If the value `37` in the root node is removed from the heap, what is the resulting heap? Use the remove method shown in class.



T. Considering the following method:

```
public void addAll(Set<String> set, String[] data) {
     assert set.isEmpty();
     for(String st : data)
          set.add(st);
}
```

`data` contains N distinct `String`s in random order.
Consider the following timing data for two different implementations of `Set`.

Implementation A
`data.length = 50,000` --- time for method `addAll` to complete: 2 seconds
`data.length = 100,000` --- time for method `addAll` to complete: 8 seconds

Implementation B
`data.length = 50,000` --- time for method `addAll` to complete: 0.5 seconds
`data.length = 100,000` --- time for method `addAll` to complete: 1 second

The internal storage container for `Set` is one of the following:
array based list, hash table, binary search tree.

Based on the timing data which internal storage container does implementation A use and which internal storage container does implementation B use?

EXTRA CREDIT (1 POINT)

If you were in class the day before the Texas - OU game I told you how proud I was of you to be in class that afternoon. And I said if you were told the user id was STUDENT then the password must be … what? Place your answer on the answer sheet.

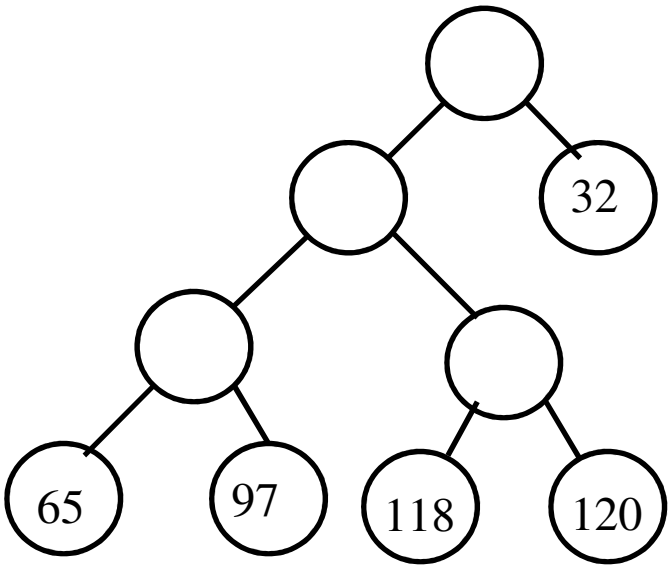
Question 2 starts on the next page.

2. (Huffman Coding - 20 points) This question has 3 parts.

Part A. (12 Points) On the Huffman Coding assignment there were two ways of building the tree from an encoded file. The first was based on the frequency of each chunk. The second was based on a binary representation of the tree itself. This question deals with a third way of building the tree, based on the codes for the chunks themselves. Consider the following example of chunks and codes.

Chunk	Code
65	000
32	1
97	001
118	010
120	011

Given the above chunks and codes the resulting Huffman code tree is:



Complete a private helper method in the Huffman Tree class that adds a new chunk to the current, partially complete tree, given the code for the chunk. Here is the `HuffmanTree` class the method is a part of.

```
public class HuffmanTree {

      private HuffNode root;

      private static class HuffNode {
            // Recall all fields of HuffNode may be accessed by code in
            // the HuffTree class. Recall the class a built in default
            // constructor.

            private int chunk;
            private int freq;
            private HuffNode leftChild;
            private HuffNode rightChild;
      }
```

Complete the following private instance method for the `HuffmanTree` class. The method adds a new chunk to the partial tree based on the chunk's Huffman code, adding nodes and links as necessary to the tree.

The frequency for all nodes (internal and leaf) shall be set to -1.

```
/* pre: root != null, all chars in code are '0' or '1',
          no chunk with code is in this tree yet.
   Add chunk to the tree adding other nodes and links as necessary.
*/
private void addChunk(int chunk, String code) {
     assert root != null
```

2.B (3 points) Complete a constructor for the HuffmanTree class. The constructor accepts a map whose keys are chunks and whose values are Strings which hold the code for a given chunk. The constructor builds a new tree out of the given chunks and codes.

Your method shall call the `addChunk` method from part A. Do not duplicate the code from part A.

```
/* pre: chunksAndCodes != null
        all chars in chunksAndCodes values are equal to '0' or '1'

   post: the HuffmanTree has been created based on chunksAndCodes
*/
public HuffmanTree(Map<Integer, String) chunksAndCodes) {
```

2.C (5 points) Huffman Trees must be complete trees. Recall a complete binary tree is one in which all nodes are leaves or have two children. Write a private instance method that returns `true` if the `HuffmanTree` is complete, `false` otherwise

Your method shall be no worse than O(h) space where h is the height of the tree. In other words you can't use arrays or other data structures equal in length to the number of elements in the tree.

```
private boolean isComplete() {
     assert root != null;
     return completeHelper(root);
}
```

Complete the following method:

```
private boolean completeHelper(HuffNode n) {
```

3. (Graphs - 15 points) Implement a method in a `Graph` class that determines if a given starting node is part of a cycle or not. Recall a cycle exists in a directed graph if there is path with two or more edges starting at a given vertex and eventually reaching that same vertex.

Consider the graph shown in question 1.P on this exam. Vertex E is part of a cycle. One example of a cycle from vertex E is E to G to A to F and back to E.

Recall the Graph class:

```
public class Graph {

    // used to indicate a vertex has not been visited and
    // that no path exists between current start vertex.
    private static final double INFINITY = Double.MAX_VALUE;

    private Map<String, Vertex> vertices;

    private static class Vertex {

        private String name;
        private List<Edge> adjacent;
        private int scratch;
        private double distance;

        public void reset() {
            distance = INFINITY;
            prev = null;
            scratch = 0;
        }
    }

    // model edge between two vertices
    private static class Edge {
        private Vertex dest;
        private double cost;
    }

    // calls the reset method on every vertex in this Graph.
    private void clearAll()

     // returns true if this Graph contains a vertex with the
     // given name.
     public boolean containsVertex(String name)
```

Complete the method on the next page.

Complete the following method.

You may use the Java `Map`, `Iterator`, `Queue`, `LinkedList`, and / or `ArrayList` classes and the `equals` method from the `String` class in addition to the `Edge` and `Vertex` classes. Do not use any other methods from the `Graph` class or `Vertex` class unless you implement them yourself.

```
/* pre: containsVertex(start) == true

   post: return true if the vertex specified by start is part
     of a cycle, false otherwise.
*/
public boolean partOfCycle(String start) {
    assert containsVertex(start);

    clearAll(); // do not change this line of code
```

4. (Hash tables, 10 points) Complete an `Iterator` class for a `Hashtable` class that uses open address hashing. The iterator shall move through the elements of the hash table based on their position in the hash table's internal array. In other words the first element returned by the iterator is the element in the array with the index closest to 0, the next element is the one with the index second closest to 0, and so forth.

Do not use any other methods in the `Hashtable` class unless you implement them yourself.

Your method must be O(1) ***space***, meaning you cannot use temporary arrays, lists, or other data structures whose size depends on the number of elements in the hash table.

Recall the `Hashtable` class:

```
public class Hashtable<E> implements Iterable<E> {

     // number of elements in this Hashtable
     private int size;

     // Elements in con that held a value at one time, but have
     // since been removed refer to EMTPY.
     private static final Object EMPTY = new Object();

     // internal container. Empty elements are either null or
     // refer to the EMPTY object.
     private E[] con;

     public Iterator<E> iterator() {
          return new HashIterator();
     }
```

Complete the following nested class inside the `Hashtable` class. Add instance variables, implement a default constructor, and implement the methods, `hasNext()`, `next()`, and `remove()`. Recall the inner class has access to the private instance variables of the outer class.

```
     private class HashIterator implements Iterator<E> {

          // add other instance variables here
          private boolean removeOK;
```

```
// complete constructor
private HashIterator() {
    removeOK = false;
```

```
// Return true if there are any more elements left in this iteration.
public boolean hasNext() {
```

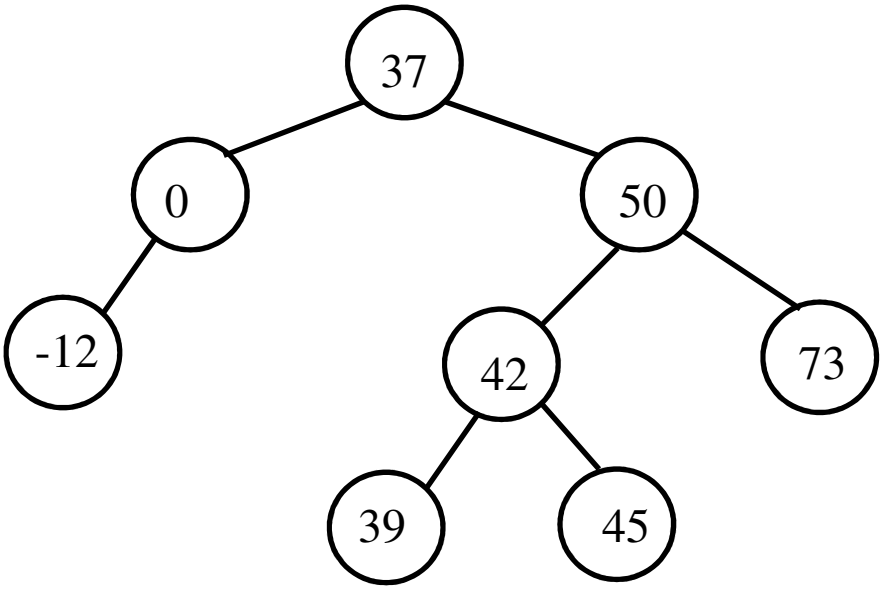
```
// Return the next element in this iteration. (also moves iterator)
public E next() {
     if(!hasNext())
          throw new NoSuchElementException();
```

```
// Remove the last element returned by this iterator.
public void remove() {
     if(!removeOK)
          throw new IllegalStateException;
```

5. (Binary Search Trees, 15 points) Implement a method that returns the median value(s) from a binary search tree.

In a set of sorted values the median is the middle element. If there are an odd number of elements the median in the middle element. If there is an even number of elements the median is the average of the two middle elements. Since we are dealing with objects in this question and cannot average them, if the number of elements is even then your method shall return the two middle elements in ascending order.

Consider the following example:



Given the above binary search tree, with an even number of elements your method would return an array of length two with 39 and 42 in it. If the node with 45 were removed there would be an odd number of elements and your method would return an array of length 1 with the value 39 in it.

Your method shall be no worse than O(h) space where h is the height of the tree. In other words you can't use arrays or other data structures equal in length to the number of elements in the tree.

Use an array of length one as a parameter to track the number of nodes that have been visited so far.

Here is the `BinarySearchTree` class:

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

     private BSTNode<E> root;
     private int size;
```

Recall the BSTNode class:

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

     public E getData()
     public BSTNode<E> getLeft()
     public BSTNode<E> getRight()
}
```

This is the kickoff method in the `BinarySearchTree` class:

```
/* pre: size() > 0
   post: return the median value(s) of this BinarySearchTree.
     If the tree size is odd return an array of length 1 with the
     median value. If the tree size is even return an array of length
     two with the two middle values in ascending order.
*/
public E[] getMedian() {
     if(size() == 0)
          throw new IllegalStateException();
     // set result to correct size
     E[] result = (size % 2 == 0) ? (E[]) (new Comparable[2])
                                  : (E[]) (new Comparable[1]);
     int[] count = new int[1];

     medianHelper(root, count, result);
     return result;
}
```

Complete the `medianHelper` method in the `BinarySearchTree` class on the next page.

```
private void medianHelper(BSTNode<E> n, int[] count, E[] result) {
```

Question 1 answer Sheet.

Name_____

A.

_____

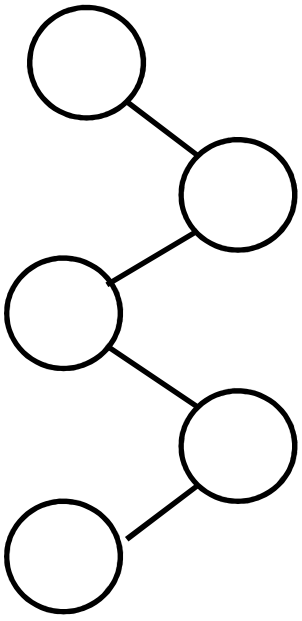
B.

_____

C.

_____

D.



E.

_____

F.

_____

G.

_____

H.

_____

I.

_____

J.

_____

K. Average case order of add method:

Order of add method if load factor near 1.0 and explanation:

_____

L.

_____

M.

_____

N.

_____

1.

O. 2.

_____

NAME_____

P.

_____

Q.

_____

R.

_____

S.

_____

implementation A internal storage container:

implementation B internal storage container:

T.

_____

EXTRA CREDIT : Password for the user id STUDENT is _____