Points off	1	2	3	4	5		Total off	Net Score

CS 307 – Final – Spring 2009

Name_____

UTEID login name _____

Instructions:

1. Please turn off your cell phones.
2. There are 5 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator on the test.
5. You may add helper methods if you wish when answering coding questions.
6. When answering coding questions assume the preconditions of the methods are met.

1. (1.5 point each, 30 points total) Short answer. Place you answers on the attached answer sheet.

For questions that ask what is the output:

- If the code contains a syntax error or other compile error, answer "Compiler error".
- If the code would result in a runtime error or exception answer "Runtime error".
- If the code results in an infinite loop answer "Infinite loop".

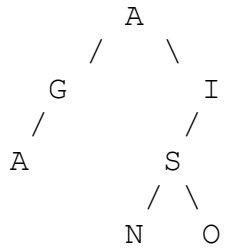
On questions that ask for the Big O of a method or algorithm, recall that when asked for Big O your answer should be the most restrictive Big O function. For example Selection Sort has an expected case Big O of O(N^2), but per the formal definition of Big O it is correct to say Selection Sort also has a Big O of O(N^3). Give the most restrictive, correct function. (Closest without going under.)

Short answer questions that refer to `ArrayLists` and `LinkedLists` are referring to the Java standard library classes unless otherwise noted.

A. The following values are inserted one at a time in the order shown into an initially empty binary search tree using the traditional, naïve insertion algorithm. Draw the resulting tree.

```
22  63  8  -20  49
```

Consider the following binary tree. `A` is the root of the tree.



B. What is the result of a pre-order traversal of the binary tree?

C. What is the result of a in-order traversal of the binary tree?

D. What is the result of a post-order traversal of the binary tree?

E. What is the result of a level-order traversal of the binary tree?

F. Is the tree shown above a binary search tree?

G. What is the best case Big O for the following method? `list` contains N elements.

```
public int count(int cutoff, LinkedList<Integer> list){
    int count = 0;
    for(int i = 0; i < list.size(); i++){
        if( list.get(i) > cutoff ){
            count++;
        }
    }
    return count;
}
```

H. What is the worst case Big O for adding N elements, one at a time to an initially empty red black tree?

I. What is output by the following code?

```
String name = "WILKINSON";
Stack<Character> st = new Stack<Character>();
for(int i = 0; i < name.length(); i++)
    st.push( name.charAt(i) );

st.pop();
st.pop();
st.push( st.peek() ); // peek returns top item without removing

for(int i = 0; i < 4; i++)
    System.out.print( st.pop() );
```

J. What is the worst case Big O of the following method? Assume the `println` method is O(1) and that `list` contains N elements.

```
public void clearPrint(double cutoff, ArrayList<Double>  list ){
    int count = 0;
    for(int i = 0; i < list.size(); i++){
        double val = list.remove( list.size() - 1 );
        if( val <= cutoff ){
            count++;
            System.out.println( val );
            System.out.println( count );
        }
    }
}
```

K. What is the Big O of the following Method? `list` contains N elements.

```
public Hashtable toHashTable(ArrayList<Integer> list){
    Hashtable result = new Hashtable();
    for(int i = 0; i < list.size(); i++){
        Integer temp = list.get(i);
        result.add( temp );
    }
    return result;
}
```

L. What are the two collision resolution schemes for hash tables that we discussed in class?

M. What does the following expression in postfix notation evaluate to?

```
2 1 + 3 2 * +
```

N. Suppose you want to encode the 26 letters of the English alphabet using a fixed length encoding scheme. (Each character uses the same number of bits.) What is the minimum number of bits per character to achieve this?

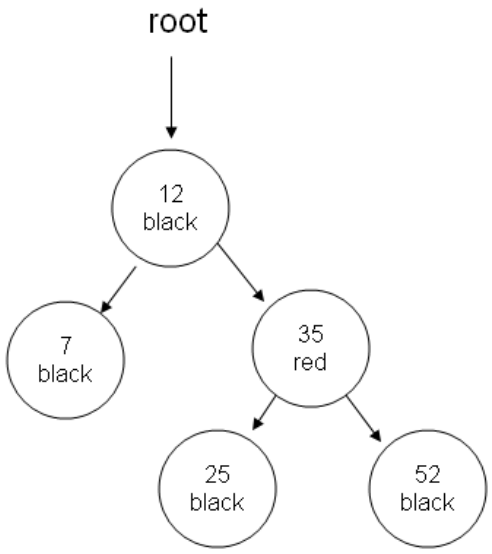
O. N elements are added to a binary search tree that is initially empty and uses the naïve insertion algorithm. What is the average case height of the resulting tree? State your answer using Big O -> O(what?)

P. You have an array of 10,000 elements. The elements in the array are unsorted. You are going to perform 1000 searches, one at a time, for elements that are present in the array without first sorting the array. Roughly how many total elements will be accessed while performing the 1000 searches?

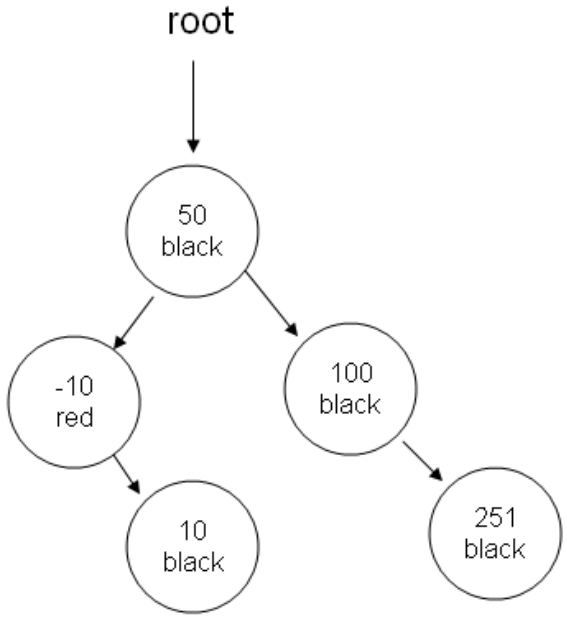
Q. Recall the requirements for a binary search tree to be a red black tree:

1. In must be a binary search tree
2. All nodes must be labeled red or black
3. The root must be black
4. Child nodes of red nodes must be black.
5. All paths from a node to any null links below that node must contain the same number of black nodes.

Is the following a Red Black tree? If not, explain why not.



R. Is the following a red black tree?. If not, explain why not.



S. You are working in Java and you need a list to store data. You will often insert elements into the middle of the list. You will only remove and access elements from the beginning and end of the list. Should you use an ArrayList, a LinkedList, or does it not matter which of those two lists your use?

T. Which data structure is best to use as the internal storage container for a regular queue to easily and efficiently (all queue operations O(1)) implement the queue?

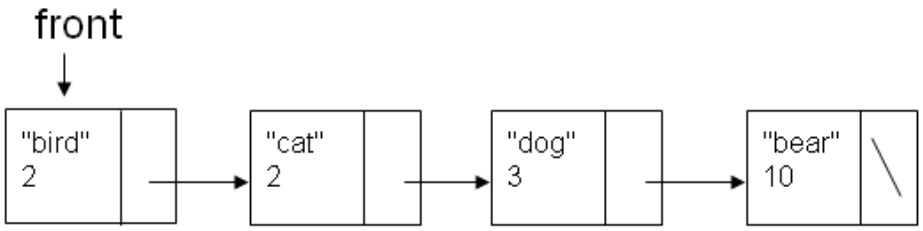
2. (Linked Lists, 20 points) Implement an enqueue method for a priority queue class that uses a singly linked list as its internal storage container. A priority queue is like a regular queue except that when values are enqueued they are inserted based on their priority where as in a regular queue values are always enqueued at the back. When a value is enqueued it is placed in front of values already in the queue that have a lower priority.

For this question priorities will be expressed as positive integers. A smaller value indicates a higher priority. Thus a priority of 1 is the highest possible priority while larger values indicate a lower priority.

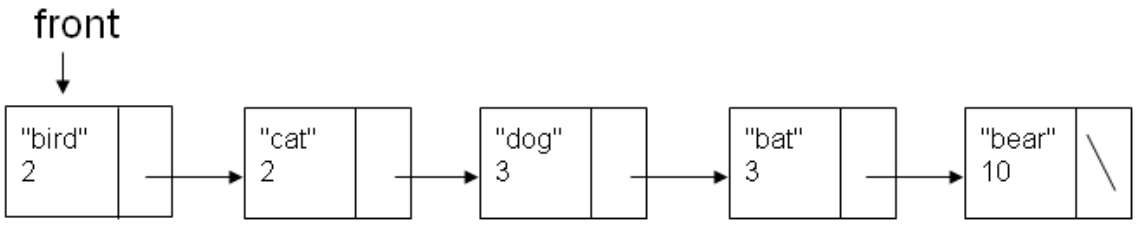
The priority queue classes uses a linked structure of nodes as its internal storage container.

- The nodes are singly linked.
- The priority queue class keeps a reference to the first node in the linked structure, named `front`.
- The first node in the linked structure is the front of the priority queue.
- The last node in the linked structure is the back of the priority queue.
- The last node in the linked structure has its next reference set to null.
- If the prioirty queue is empty the reference to the front node is set to null.
- The priority queue does not have an instance variable for its size.
- The priority achieves genericty by using Java's generics.

Consider the following priority queue and its internal data structure. The priorty queue in this example contains strings. The priority for each value is also shown.



In this example assume **"bat"** is enqueued with a priority of **3**. The resulting priority queue and internal data structure is shown below.



Imporant: Your method must be O(1) space meaning you can not use an auxillaury native array, ArrayList, or other data structure.

Here is the `Node` class the priority queue class uses:

```
public class Node<E>{
    public Node(E value, int priority, Node<E> next);

    public Node<E> getNext();
    public int getPriority();
    public E getValue();

    public void setNext(Node<E> newNext);
}
```

Here is the `PriortityQueue` class.

```
public class PriorityQueue<E>{

    // Reference to the front node of the linked list.
    // If this priority queue is empty front == null
    private Node<E> front;

    // complete the following method:
    // pre: value != null, priority > 0;
    public void enqueue(E value, int priority){
        assert value != null && priority > 0
```

// More room on next page if needed

3. (Implementing Data Structure, 15 points) Write a method to add a value to a hash table. Recall that a hash table is a data structure that uses an array as its internal storage container. Items are added to the array based on the integer generated by a hash function. A hash function produces an integer based on properties of the object. In Java hash functions are encapsulated via the `hashCode` method in the `Object` class. Most classes override the `hashCode` method.

`Object` are added to this hash table via the following algorithm:

1. call the object's `hashCode` method to obtain an int, x
2. modulus x by the length of the hash table's internal native array to obtain an index, i
3. go to index i in the hash table's internal native array
4. Note there is a special case. The hash table you are implementing maintains the set property so if the object being added is already present the hash table is unchanged.
5. if that element is null or equal to the `NOTHING` object, place the `Object` in that element
6. if that element is not null a collision has occurred.
7. In this hash table collisions are resolved via *linear probing*. Check the element at index i + 1
8. If the element at i + 1 is null or equal to the `NOTHING` object, add the item at that element, if not continue to check elements until an open spot is found. The distance from the original spot increases linearly : i + 1, i + 2, i + 3, and so forth. If the end of the array is reached then wraparound.
9. When the hash table's load factor is reaches the load factor limit its internal storage container is resized and items are rehashed and added to the new internal storage container.

```
public class HashTable{
      // The NOTHING object. Values that are removed from the
      // hash table are replaced with a reference to this
      // object instead of being set to null
      private static final Object NOTHING = new Object();

      // storage container for items in this HashTable
      private Object[] con;

      // number of items in this HashTable
      private int size

      // load factor limit. When the load factor (number of elements in
      // the has table divided by the capacity of the internal array)
      // of the hash table is greater than or equal to this value the
      // resizeAndRehash method must be called.
     private double loadFactorLimit;

      // method to resize internal storage container and place values
      // in the new storage container
      // DO NOT WRITE THIS METHOD! You may call it as if it has already
      // been completed.
     private void resizeAndRehash()

      // complete the following method.
      // Write your answer on the next page
      /*    pre: obj != null
            post: Add obj to this hash table. Return true if obj was not
            already present in this hash table, false otherwise.
      */
      public boolean add(Object obj)
}
```

You will have to use the `hashCode` method from the `Object` class:

```
public int hashCode()
      Returns the hash code value for this Object.
```

Complete the following instance method from the hash table class.

```
/*    pre: obj != null
      post: Add obj to this hash table. Return true if obj was not
already    present in this hash table, false otherwise.
*/
public boolean add(Object obj)
```

4. (Binary Search Trees, 20 points) This questions has two parts. Each parts is worth 10 points. In each part you will write an instance method for the following binary search tree class.

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

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

Here is the public interface of the `BSTNode` class.

```
public class BSTNode<E extends Comparable<E>> {
    public BSTNode()
    public BSTNode(E initValue)
    public BSTNode(E initValue, BSTNode<E> l, BSTNode<E> r)

    public E getValue()
    public BSTNode<E> getLeft()
    public BSTNode<E> getRight()

    public void setValue(E theNewValue)
    public void setLeft(BSTNode<E> theNewLeft)
    public void setRight(BSTNode<E> theNewRight)
}
```

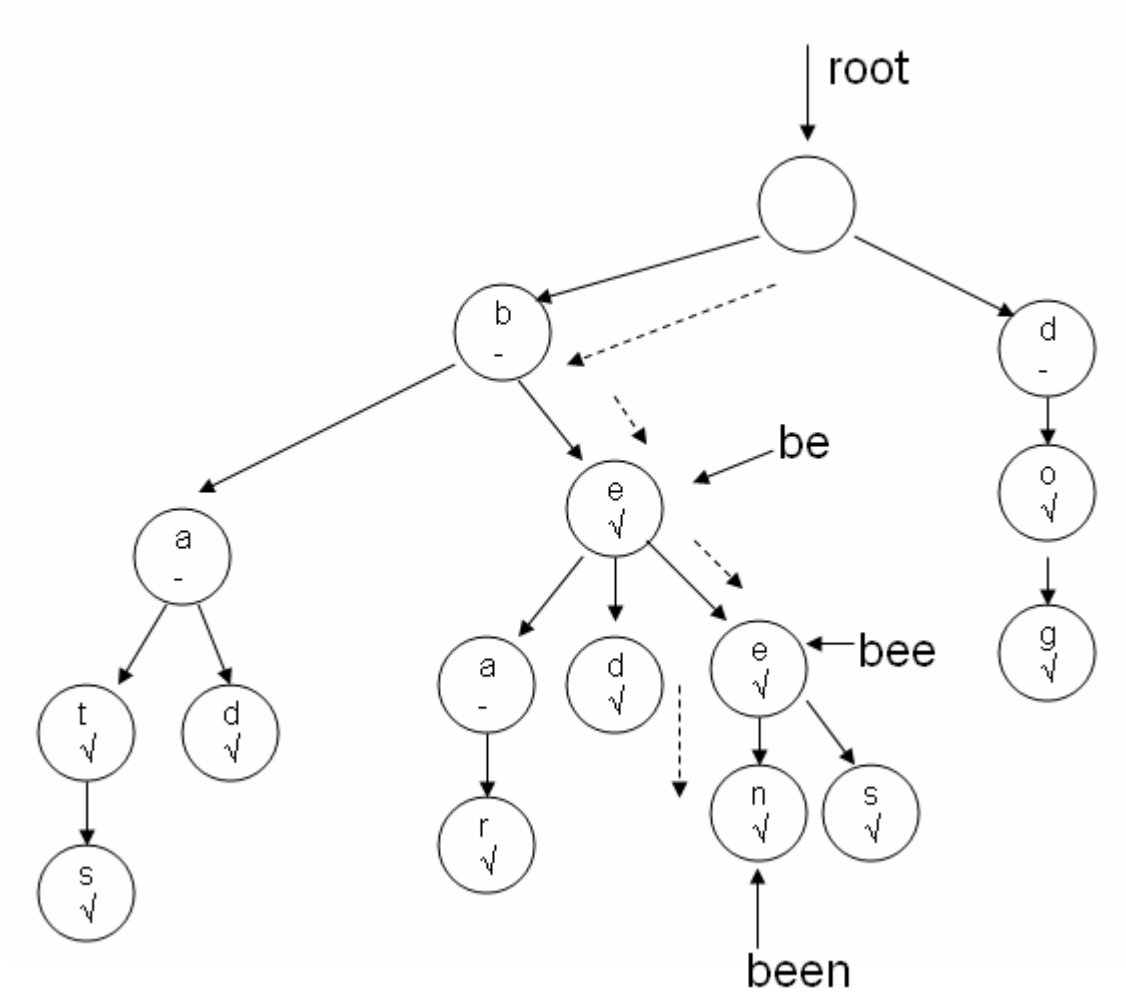
Part A: Write a method that returns the number of leaf nodes in the binary search tree. Recall a leaf node is one that has 0 children. This is an instance method in the BinarySearchTree class.

```
// pre: none
// post: return the number of leaf nodes in this BianrySearchTree
public int numLeaves(){
```

Part B: Write a method to remove the maximum value in the binary search tree. Your method shall be no worse than O(dm) space where dm is the depth of the node that contains the maximum value in the binary search tree. Recall the max value in a binary search tree is located in the rightmost node that can be reached starting at the root node of the tree.

```
/* pre: size() > 0
   post: remove the maximum value in this BinarySearchTree.
   Size is decremented.
*/
public void removeMax(){
    assert size() > 0;
```

5. (New data structures, 15 points) A *trie* is a tree based data structure that stores strings and supports fast look up as well as other operations. The root node of the tree does not contain any values. Descendants of the root node contain a single character and a boolean to indicate if the path from the root to that node is a valid word. Here is a very small example of a trie.



Strings are built up by starting at the root and descending through the tree. A dash in a node indicates the string built up along that path is not a valid word, although it is a prefix to other valid words. A check indicates the string built up along that path is a valid word.

The dashed lines shown how the word "been" is built. The path to "be" and "bee" are also shown. The words present in the above trie are: bat, bats, bad, be, bear, bed, bee, been, bees, do, and dog.

Tries are not binary trees. The number of possible child nodes is equal to the number of letters in the alphabet that is being used to form words. In this question nodes will have between 0 and 26 children. Child nodes are identified by the letter they would contain.

Complete a method in the Trie class that returns true if a given word is in the Trie.

```
public boolean wordIsPresent(String word)
```

If the variable t is the Trie shown above, here are the results of some calls to `wordIsPresent`.

`wordIsPresent("b")` returns false
`wordIsPresent("be")` returns true
`wordIsPresent("bee")` returns true
`wordIsPresent("bees")` returns true
`wordIsPresent("abra")` returns false
`wordIsPresent("dogs")` returns false

Here is the Trie class:

```
public class Trie{

    // root is never null, root does not contain a valid character
    private TNode root;

    // other methods not shown
}
```

And here are the methods from the TNode class you can use.

```
public class TNode{

     // returns the character in this node. All
     // characters will be a lower case letter, 'a' to 'z'
     public char getChar()

     // Returns true if the word formed by the path from the root
     // to this node is a word in this Trie, false otherwise.
     public boolean isWord()

     // pre: ch is lower case letter, 'a' to 'z'
     // Returns the reference to the child node that contains the
     // character ch or null if the child does not exist.
     public TNode getChild(char ch)

}
```

Complete the following method on the next page. This is an instance method of the Trie class

```
/* pre: The parameter pre is not null and only contains lower
   case letters.
   post: returns true if the given word is in this Trie, false
   otherwise.
*/
public boolean wordIsPresent(String word){
```

```
/* pre: The parameter pre is not null and only contains lower
   case letters.
   post: returns true if the given word is in this Trie, false
   otherwise.
*/
public boolean wordIsPresent(String word){
     // you are not required to check the precondition
```

Name:_____

Answer sheet for question 1, short answer questions. **Put answers next to the letter.**

A.

B.

C.

D.

E.

F.

G.

H.

I.

J.

K.

L.

M.

N.

O.

P.

Q.

R.

S.

T.