Points off	1	2	3	4	5		Total off	Net Score

CS 307 – Final – Fall 2008

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".

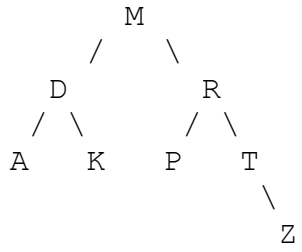
On questions that ask for the Big O of a method or algorithm, recall that when asked for Big O your answer should be the most restrictive Big O function. For example Selection Sort has an expected case Big O of O(N^2), but per the formal definition of Big O it is correct to say Selection Sort also has 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.

```
17   -12  5    0    20   15
```

Consider the following binary tree. `M` 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 height of the tree shown above?

H. What is the worst case Big O for adding N elements, one at a time to an initially empty binary search tree using the traditional, naïve insertion algorithm?

I. What is the Big O of the following method? Assume `Math.random` is O(1). `list` is initially empty.

```
public void randFill(int N, LinkedList<Double>  list ){
     assert list.size() == 0;

     for(int i = 0; i < N; i++)
          list.add(Math.random());

     double total = 0.0;
     for(int i = 0; i < list.size(); i++)
          total += list.get(i);
}
```

J. What is the Big O of the following method?

```
public ArrayList<Integer> createAndFill(int N){
     ArrayList<Integer> result = new ArrayList<Integer>();
     for(int i = 0; i < N; i++)
          result.add(i);
     return result;
}
```

K. An `UnsortedSet class` uses a Java `ArrayList` as its internal storage container. The `UnsortedSet` contains N `String`s. The `UnsortedSet` includes a method to remove a given `String` from the `UnsortedSet` if it is present.
What is the expected Big O of the `UnsortedSet`'s `remove` method?

L. What is the Big O of the following method? `list` initially contains N elements.

```
public void keep5(LinkedList<Integer> list){
     while( list.size() != 5 ){
          list.remove(0); // position based removal
     }
}
```

M. What is the worst case height of a Red Black tree that contains N elements? Express your answer using Big O notation. (O(what?))

N. What is the worst case Big O of the following method? The `LinkedList` initially contains N items.

```
public int numBigger(LinkedList<Integer> list, int target){
     int total = 0;
     Iterator<Integer> it = list.iterator();
     while( it.hasNext() ){
          int temp = it.next();
          if( temp > target ){
               total++;
          }
     }
     return total;
}
```

O. What is the running time of the following constructor for an array based list class?
N = `initialCapacity`.

```
public class ArrayBasedList{
     private int size;
     private Object[] container;

     // pre: initialCapacity >= 0;
     public ArrayBasedList(int initialCapacity){
          assert initialCapacity >= 0;
          size = 0;
          container = new Object[initialCapacity];
     }

     // other methods not shown
}
```

P. You are using a sorting algorithm that uses the merge sort algorithm. When sorting 1,000,000 values it takes the method 1 second to sort the data. How long do you expect it to take to sort 4,000,000 values? Recall $\log_2 1{,}000{,}000 \sim= 20$.

Q. In class we discussed how to resolve collisions in hash tables. In open address hashing the collision resolution technique involves probing for an open spot. What is the other common collision resolution technique and *briefly* explain how it works.

R. *Briefly* explain the primary reason to store data in sorted order.

S. You are implementing a stack with an array based list. The array based list maintains extra capacity in its internal storage container, an array. The elements of the list are stored from index `0` to index `size - 1` in the array. Where should the top of the stack so that all traditional stack operations are as efficient as possible. *Briefly* explain your answer.

T. What does the following postfix expression evaluate to?

```
3 2 + -2 *
```

2. (Binary Trees, 20 points) We define the *balance factor* of a binary search tree as the ratio between its height and its minimum possible height. (actual height / min possible height) Write an instance method in a binary search tree class that determines the balance factor of that binary search tree. Bad News: this binary search tree does not have a completed height method and it does not track its size. (Darn lazy programmers!)

Recall the height of tree is the path length (number of links) from the root to the deepest leaf.

When writing your solution you cannot use other methods in the binary search tree class. You cannot use other methods or classes from the Java standard library. You cannot use auxiliary arrays. You can use the BSTNode class.

Hint: Create helper methods to find the height of the tree, the size of the tree, and the minimum possible height of the tree given the size.

Here is the binary search tree class:

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

     private BSTNode<E> root;
```

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)
}
```

Complete the following method.

```
// pre: There are at least 2 values in this binary search tree.
//     Thus height will be greater than 0.
// post: Return the balance factor for this binary search tree.
//   This binary search tree is not altered as a result of this
//   method.
public double balanceFactor(){

// Complete this method and any helper methods on the
// following pages:
```

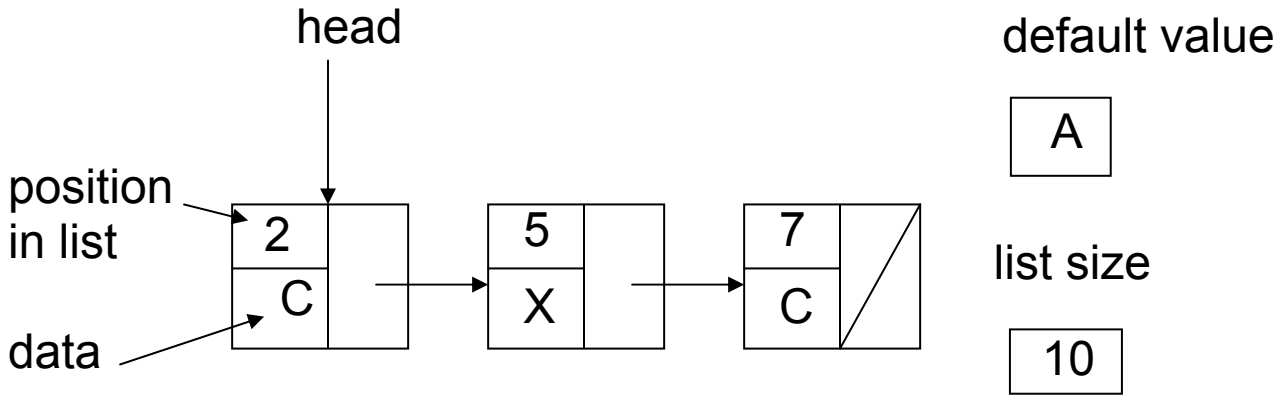
```
// pre: There are at least 2 values in this binary search tree.
//     Thus height will be greater than 0.
// post: Return the balance factor for this binary search tree.
//     This binary search tree is not altered as a result of this
//     method.
public double balanceFactor(){
```

3. (Linked List, 20 points) A *sparse list* is a list in which most elements are 0 (or some other default value). There are only a few elements in the list not equal to the deault value. It may be possible to save time and space implementing sparse list by only storing the elements of the list that are not 0. If this is done using a linked structure of nodes (as in a linked list) then the position in the list is no longer tied implicitly to the position of the node in the linked sturcture of nodes.

The `SparseLinkedList` class for this question has the following properties:

- Values are stored in singly linked nodes.
- The list maintains a default value. Any elements of the list that are not explicty store in the linked structure are equal to this deafault value. The default value is not stored in the linked structure of nodes.
- Each node stores one value and the position of that valuein the list.
- The relative order of nodes in the linked structure is the same as the relative order of the values in the list not equal to the default value.
- The list uses 0 based indexing, so the first element of the list is at position 0.
- The class has a reference to the first node in the linked structure named `head`.
- The class does not have a reference to the last node in the list.
- The class does not store the number of nodes in the linked structure, only the size of the list being represented.
- The last node in the linked structure has its next reference set to `null`.
- When there are no values in the list not equal to the deault value, `head` is set to `null`.

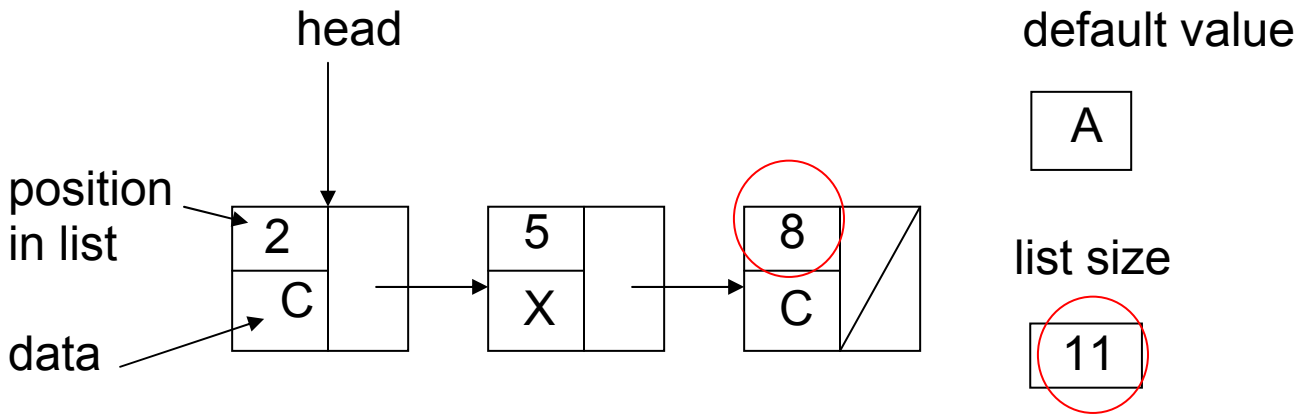
Consider the following example. The default value for this list is `A` and the list size is 10.



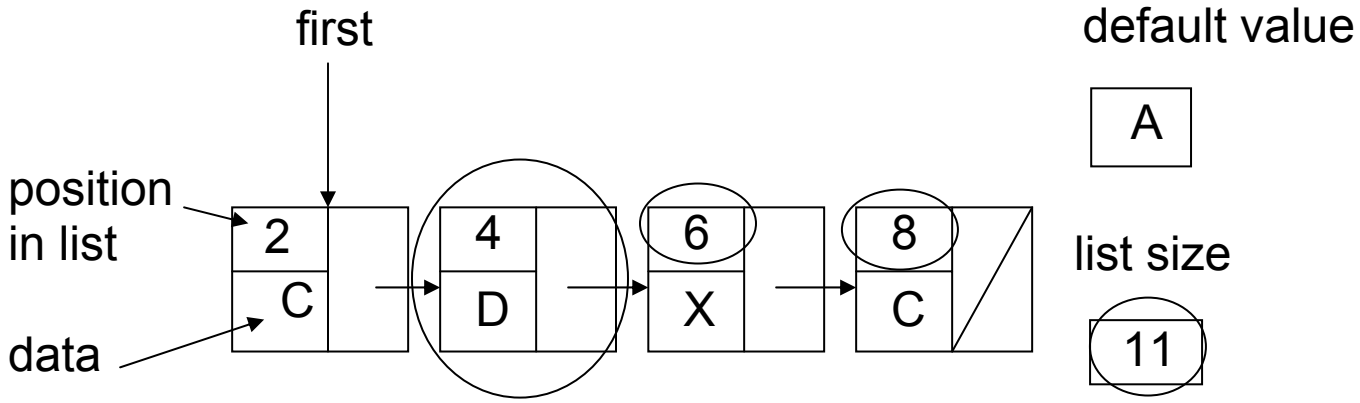
The above is a picture of the `SparseLinkedList` object that represents the following list:
`[A, A, C, A, A, X, A, C, A, A]`

If an `A` was inserted into the list at position at 6 the list would become:
`[A, A, C, A, A, X, A, C, A, A]`

The position of the second C in the list is updated to be 8 and the list size is updated to be 11. The next picture shows the changes to the `SparseLinkedList` after inserting an A at position 6.



If instead a `D` was inserted into the list at position 4 the list would become: `[A, A, C, A, D, A, X, A, C, A, A]` and the `SparseLinkedList` would look like this:



Write an `insert` method for the `SparseLinkedList` class. You cannot use any other methods from the `SparseLinkedList` class. You solution must be O(1) ***space*** which means you cannot use an auxiliary `array` or other data structure to complete the method.

Here is the singly linked node class the `SparseLinkedList` class uses. `position` refers to the position in the list of that node's value.

```
public class Node<E>{
      public Node(E obj, Node<E> next, int position)

      public E getData()
      public Node<E> getNext()
     public int getPosition();

      public void setData(E obj)
      public void setNext(Node<E> next)
      public void setPosition(int newPosition)
}
```

Here is the `SparseLinkedList` class.

```
public class SparseLinkedList<E>{

     private Node<E> head;
     private E defaultValue;
     private int listSize;

     //Complete the following method:
     // pre: obj != null, 0 <= pos <= size() (of list)
     // post: obj added to this list at position pos
     public void insert(E obj, int pos){
```

4. (Implementing data structures, 15 points) Write a method to remove a value from 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 and that most classes override.

`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 to obtain an index, i
3. go to index i in the hash tables native array
4. if that element is null or equal to the `NOTHING` object, place the `Object` in that element
5. if that element is not null a collision has occurred.
6. In this hash table collisions are resolved via *quadratic probing*. Check the element at index i + 1
7. 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 goes up quadratically: 1, 2, 4, 8, 16, 32, 64, and so forth. If the end of the array is reached then wraparound.
8. When the hash table's load factor is reached its internal storage container is resized and items are rehashed and placed in the proper spot.

When removing a value from the hash table you must determine if the value is present or not. If it is present replace that object with the `Nothing` object. You cannot use any other methods from the `HashTable` class. You solution must be O(1) ***space*** which means you cannot use an auxiliary `array` or other data structure to complete the method.

```
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

      // complete the following method.
      // Write your answer on the next page

      /*    pre: item != null
            post: If obj is present in the hash table this method removes
            it, decrements size, and returns true. If obj is not present
            the method returns false and the hash table is unchanged.
      */

      public boolean remove(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 method from the hash table class.

```
/*    pre: item != null
      post: If obj is present in the hash table this method removes it,
      decrements size, and returns true. If obj is not present the method
      returns false and the hash table is unchanged.
*/
public boolean remove(Object obj)
```

5. (Using data structures, implementing data structures 15 points) Assume you have implemented a `Stack` with the following methods:

```
public class Stack<E>{
     // create empty Stack
     public Stack()

     // add item to Stack
     public void push(E item)

     //get top item
     public E top()

     //remove top item
     public E pop()

     //return true if no items in this Stack, false otherwise
     public boolean isEmpty()
}
```

Assume you are now implementing a `Queue` class and you must (for unexplained reasons) use stacks as your internal storage containers. Implement a complete `Queue` class with the traditional queue methods: `enqueue`, `dequeue`, `front`, and `isEmpty`. **The only instance variables you may use are 2 `Stack` objects. You may not use any arrays or other data structures as instance or local variables.**

```
public class Queue<E>{

     private Stack<E> stack1;
     private Stack<E> stack2;

     public Queue(){
          stack1 = new Stack<E>();
          stack2 = new Stack<E>();
     }
```

```
// more room on next page for Queue class
```

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.