Exam Number:

Points off	1	2	3	4	5	Total off	Net Score

CS 307 – Midterm 2 – Fall 2009

Name_____

UTEID login name _____

TA's Name: Oswaldo Rashid Swati (Circle One)

Instructions:

1. Please turn off your cell phones and other electronic devices.
2. There are 5 questions on this test.
3. You have 2 hours to complete the test.
4. You may not use a calculator on the test.
5. When code is required, write Java code.
6. When writing methods, assume the preconditions of the method are met. Do not write code to check the preconditions.
7. On coding question you may add helper methods if you wish.
8. After completing the test please turn it in to one of the test proctors and show them your UTID.

1. (2 points each, 30 points total) Short answer. Place you answers on the attached answer sheet.

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

Recall that 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(5)`?

```
public int a(int n){
    if(n <= 1)
        return 3;
    else
        return a(n - 2) + n;
}
```

B. What is returned by the method call `b("stack")`? Recall the `substring` method:

`public String` **`substring`**`(int beginIndex, int endIndex)`

> Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

```
public String b(String s){
    if(s.length() <= 2)
        return s;
    else {
        int last = s.length() - 1;
        return s.charAt( s.length() - 1 ) + b(s.substring(1, last))
            + s.charAt(0);
    }
}
```

C. What is output when method `c` is called?

```
public String subC(int n){
    if(n <= 0)
        return "*";
    else
        return "*" + subC(n - 1) + "*" + subC(n - 1);
}

public void c(){
    int n = 3;
    String result = subC(n);
    System.out.println( result.length() );
}
```

D. What is output when method `d` is called?

```
public void subD(int n){
    if(n <= 2)
        System.out.print(n);
    else{
        subD(n - n / 2);
        System.out.print(n);
    }
}

public void d(){
    subD(7);
}
```

E. What is the Big O of method `e`? N = `list.length`.

```
public double e(double[] list){
    double total = 0;
    for(int i = 0; i < list.length; i++)
        total += list[i];

    for(int j = 1; j < list.length; j *= 2)
        total = total + list[j];

    return total;
}
```

F. What is the best case Big O of method `f`? N = `list.length`. Method `subF` is O(N) where N equals `list.length`.

```
public int f(int[] list){
    int total = 0;
    for(int i = 0; i < list.length; i++){
        if( list[i] % 2 == 0 || list[1] % 7 == 0 )
            total += subF(list);
    }
    return total;
}
```

G. What is the best case Big O of method `g`? N = `list.length`.

```
public int g(int[] list){
    int total = 0;
    for(int i = 0; i < list.length; i++)
        for(int j = i; j < list.length; j++)
            if(list[i] == list[j])
                total++;
    return total;
}
```

H. What is the Big O of method `h`? `mat` is a square matrix and N = `mat.length`. Assume the precondition of `h` is met.

```
// pre: mat is a square matrix, mat.length >= 3
public int h(int[][] mat){
    int total = 0;
    int middle = mat.length / 2;
    for(int r = middle - 1; r <= middle + 1; r++){
        for(int c = 0; c < mat[0].length; c++){
            total += mat[r][c];
        }
    }
    return total;
}
```

I. What is the Big O of method `i`? N = `org.length`. The `ArrayList` constructor is O(N).

```
public ArrayList<String> i(String[] org){
    ArrayList<String> result = new ArrayList<String>(org.length);
    for(int i = 0; i < org.length; i++)
        result.add(0, org[i]); // 0 is the position to insert in the list
    return result;
}
```

J. What is the Big O of method `j`? N = `org.length`. The `LinkedList` constructor is O(1).

```
public LinkedList<String> j(String[] org){
    LinkedList< String > result = new LinkedList<String>();
    for(int i = 0; i < org.length; i++)
        result.add(0, org[i]); // 0 is the position to insert in the list
    return result;
}
```

K. What is the worst case Big O of the quicksort? Under what conditions does this occur?

L. Consider the following code segment

```
int n = 125000;
Stopwatch s = new Stopwatch();
for(int i = 0; i < 5; i++){
    int[] list = new int[n];
    // code to fill list with random values
    s.start();
    mystery(list);
    s.stop();
    System.out.println(s);
    n *= 2;
}
```

When run the following times are output:

```
 2.00 seconds
 4.24 seconds
 8.95 seconds
18.84 seconds
19.78 seconds
```

Based on these times what is the most likely Big O for method `mystery` when N = `list.length`?

M. A method is $O(N^3)$. When N = 2000 the method takes 3 seconds to complete. What is the expected time to complete when N = 4000?

N. A method is $O(N^2)$. When N = 100000 the method takes 10 seconds to complete. Given 6 seconds how many pieces of data can the method process?

O. Consider the following methods from the Java `ArrayList` class.

Method Summary	
`void`	**`add`**`(E o)` Appends the specified element to the end of this list.
`void`	**`add`**`(int index, E element)` Inserts the specified element at the specified position in this list.
`E`	**`get`**`(int index)` Returns the element at the specified position in this list.
`E`	**`remove`**`(int index)` Removes the element at the specified position in this list. Return the element that is removed from the list.
`E`	**`set`**`(int index, E element)` Replaces the element at the specified position in this list with the specified element. Returns the element that was previously at index.
`int`	**`size`**`()` Returns the number of elements in this list.

What is output when method `methodO` is called?

```
public void methodO(){
    ArrayList<Integer> ls = new ArrayList<Integer>();

    ls.add(7);
    ls.add(ls.size());
    ls.add(0, 5);
    ls.add(3);
    ls.add(ls.remove(1));

    for(int x : ls)
        System.out.print(x);
}
```

2. (Using data structures, 17 points) Write a method for the `AbstractSet` class from assignment 8 that determines if the calling object and another `ISet` object passed as a parameter are *disjoint* sets. Two sets are disjoint if they share no elements in common. (In other words their intersection is the empty set.) Complete a method that returns `true` if the calling object and the parameter are disjoint, false otherwise.

- The only method you can use from `ISet` is the `iterator` method. If you want to use any other methods from `ISet` you must implement them yourself.
- Recall, the `AbstractSet` class does not have any instance variables.
- Do not make any explicit references to `UnsortedSet` or `SortedSet`.
- You can use all of the methods from the `Iterator` interface.
- Your method must be O(1) space, meaning you cannot use temporary arrays, lists, or other data structures besides the `iterator` objects which are O(1) space.
- Neither `ISet` is altered as a result of this method call.

Examples:

```
(1, 2).areDisjoint( (2, 3) ) -> false
(1, 2).areDisjoint( (2, 1) ) -> false
(1, 2, 3, 4, 5).areDisjoint( () ) -> true // other is empty set
().areDisjoint( () ) -> true // both sets are empty set
(4, 1, 2, 15).areDisjoint( (12, 5, -1, 7) ) -> true
```

`Iterator<E>` **`iterator`**`()`
Returns an iterator over the elements in this set.

Recall the methods from the `Iterator` interface:

`boolean` **`hasNext`**`()`
Returns `true` if the iteration has more elements.

`E` **`next`**`()`
Returns the next element in the iteration.

`void` **`remove`**`()`
Removes from the underlying collection the last element returned by the iterator

Complete the following method on the next page. Recall this method is part of the `AbstractSet` class.

```
// pre: other != null
// post: return true if this ISet and other are disjoint,
// false otherwise
public boolean areDisjoint(ISet other){
```

```
public abstract class AbstractSet<E> implements ISet<E> {

// pre: other != null
// post: return true if this ISet and other are disjoint,
// false otherwise
public boolean areDisjoint(ISet<E> other){
    assert other != null;
```

3. (Implementing data structures, 18 points). Write an instance method for the `GenericList` class we developed in lecture that returns a reversed sub list based on two integer parameters.

- The `GenericList` is generic based on Java's inheritance requirement and polymorphism, not the Java generic syntax.
- `GenericList`'s internal storage is a native array of `Objects`. There may be extra capacity in the array.
- The elements of the list use 0 based indexing. The position of an element in the list is equal to its index in the array.
- You may only use the `GenericList` default constructor and the `size` method. If you want to use any other methods from `GenericList` you must implement them yourself.
- Recall that since the method you are writing is part of the `GenericList` class you have access to all `GenricLists`' private instance variables.
- You may not use any other Java classes except `GenericList` and native arrays.

Complete the following method in the `GenericList` class:

```
/*   pre: 0 <= start <= stop <= size()
     post: return a new GenericList with the elements from start
     inclusive to stop exclusive in reverse order. The size of
     the returned list = stop - start.
     This GenericList is not  altered as a result of this method
     call.
*/
public GenericList getReversedSubList(int start, int stop){
```

Examples of results of `getReversedSubList`:

```
[A, C, D, A, B]. getReversedSubList (1, 3) returns [D, C]
[A, C, D, A, B]. getReversedSubList (0, 5) returns [B, A, D, C, A]
[A, C, D, A, B]. getReversedSubList (1, 1) returns [] an empty list
[A, C, D, A, B]. getReversedSubList (3, 4) returns [A]
[A, C, D, A, B]. getReversedSubList (0, 4) returns [A, D, C, A]
```

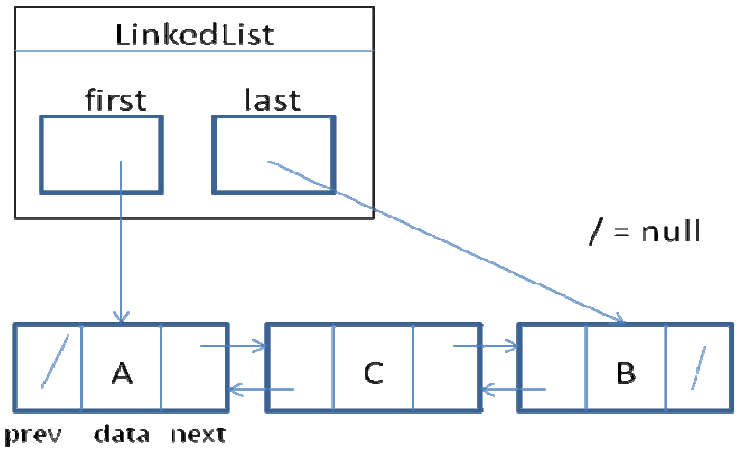
```
public class GenericList {

    private Object[] container;
    private int listSize;

    public GenericList() {
        container = new Object[10];
        listSize = 0;
    }

    public int size() {
        return listSize;
    }

// complete the getReversedSubList method on the next page
```

```
/*   pre: 0 <= start <= stop <= size()
     post: return a new GenericList with the elements from start
     inclusive to stop exclusive in reverse order. The size of
     the returned list = stop - start.
     This GenericList is not  altered as a result of this method
     call.
*/
public GenericList getReversedSubList(int start, int stop){
    assert 0 <= start && start <= stop && stop <= size();
```

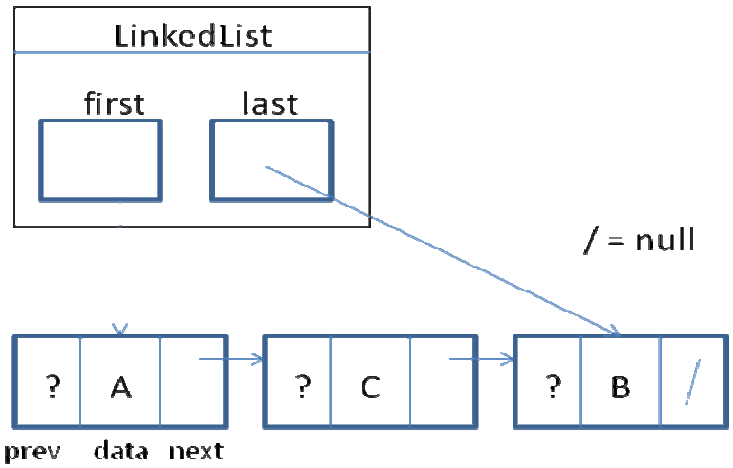
4. (Linked lists, 18 points) Write a method that determines if all of the previous references in a double linked list are correctly set. The doubly linked list has the following properties:

- The list uses doubly linked nodes that store one piece of data, a reference to the next node in the list, and a reference to the previous node in the list.
- The list maintains references to the first and last nodes in the list
- If the list is empty the references to the first and last nodes are set to `null`.
- The list does not maintain a size instance variable.
- The first node's previous reference should be set to `null`. The last node's next reference is set to `null`.

For example the list `[A, C, B]` should be stored as follows:



For this question you know each node's next reference is set correctly and the last node's next reference (if it exists) is set to null. You also know first and last are set correctly. However you are unsure if the previous references are set correctly. Write a private method for the `LinkedList` class that returns true if the previous references for the doubly linked nodes are set correctly, false otherwise. (Note it is possible the size of the list is zero in which case the method shall return true.)



Note, you are not required to repair the links in this question, just determine if they are correct or not.

Here is the Node class:

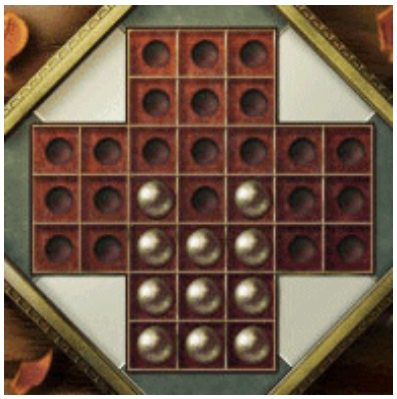
```
public class Node<E> {
     public Node(E value, Node<E> next, Node<E> prev)

     public E getData()
     public Node<E> getNext()
     public Node<E> getPrev()

     public void setValue(e value)
     public void setNext(Node<E> next)
     public void setPrev(Node<E> prev)
}
```

Here is the linked list class:

```
public class LinkedList<E> {
    private Node<E> first;
    private Node<E> last;

    // pre: all next references set correctly, first and last
    // set correctly.
    // post: return true if all the prev references are set
    // correctly, false otherwise
    private boolean prevsSetCorrectly() {
        // assume preconditions are met, do not check them
```

```
// more room on next page if necessary
```

5. (Recursion, 17 points) Write a method to determine if it possible to solve a marble jumping puzzle or not. The puzzle uses a board such as this:



The board is represented as rectangular 2d array of chars. Open spaces will indicated with an `o`. Marbles will be indicated with an `m`. Cells that cannot be moved into will be represented with an `x`. So, for example the above board will be represented with a 7 by 7 array of chars:

0	1	2	3	4	5	6
x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	**o**	o	o	o	o
o	o	**m**	o	m	o	o
o	o	**m**	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

Note this is simply one example. The board size, arrangement, and initial marble position will vary.

In the marble jumping puzzle marbles can jump over adjacent marbles that are located up, down, left, or right of the marble as long as there is an open spot on the other side of the marble being jumped over. When a marble is jumped it is removed from the board. So in the example above one of the legal moves would be to take bolded marble and move it over the marble above it leading to this configuration where the marble that was jumped over has been removed.

x	x	o	o	o	x	x
x	x	o	o	o	x	x
o	o	***m***	o	o	o	o
o	o	**o**	o	m	o	o
o	o	**o**	m	m	o	o
x	x	m	m	m	x	x
x	x	m	m	m	x	x

So a move consists of three changes:

1. Picking up the source marble from its spot.
2. Moving the source marble to its destination.
3. Removing the marble the source marble passed over from the board.

Assume a method that determines all of the legal moves for a given board already exists. The method returns an `ArrayList` of `LegalMove` objects (see class description below). You can use the methods from the `ArrayList` class listed in question 1.O and the for each loop.

```
// pre: board != null, board is rectangular
// post: returns the legal moves on board. If there are no legal
// moves an ArrayList of size 0 is returned. board is not altered.
public ArrayList<LegalMove> getLegalMoves(char[][] board)
// DO NOT COMPLETE THIS METHOD. USE IT ASSUMING IT WORKS AS DESCRIBED.
```

Here is the `LegalMove` class:

```
public class LegalMove {

      public int sourceRow() // returns row of marble to move
      public int sourceCol() // returns column of marble to move

      public int destRow() // returns row to move marble to
      public int destCol() // returns column to move marble to

      // returns row of marble removed if this move is made
      public int removedRow()
      // returns column of marble removed if this move is made
      public int removedCol()
}
```

Further, assume there is a method that returns the number of marbles on the board:

```
// pre: board != null, board is rectangular
// post: return the number of m's in board
public int numMarblesOnBoard(char[][] board) {
// DO NOT COMPLETE THIS METHOD. USE IT ASSUMING IT WORKS AS DESCRIBED.
```

Complete the following method:

```
// pre: board != null, board is rectangular, board only contains
//  the characters x, m, and o. Initially there are at least 2 m's.
// post: return true if the puzzle represented can be solved.
// In other words, a series of legal moves can be made so that
// only one marble remains.
public boolean canBeSolved(char[][] board){
      // assume preconditions are met, do not check them
```

Complete this method on the next page.

```
// pre: board != null, board is rectangular, board only contains
//  the characters x, m, and o. Initially there are at least 2 m's.
// post: return true if the puzzle represented can be solved.
// In other words, a series of legal moves can be made so that
// only one marble remains.
public boolean canBeSolved(char[][] board){
     // assume preconditions are met, do not check them
```

Name:_____

TAs name: Oswaldo Rashid Swati (Circle One)

Answer sheet for question 1, short answer questions

A. _____ I. _____

B. _____ J. _____

C. _____ K. _____

D. _____ L. _____

E. _____ M. _____

F. _____ N. _____

G. _____ O. _____

H. _____