Questions 1-6 and 9 are based on the class `SinglyLinkedList`, which begins as follows.

```java
public class SinglyLinkedList {
    private static class Node {
        Object data;
        Node   next = null;

        private Node ( Object data ) {  // Only constructor provided.
            this.data = data;
        }

        // There is no two-arg. constructor
    }

    // No list constructor is needed; all fields initialized to empty list state.
    // This list does NOT use a dummy header and is not a circular list.
    private Node head = null; // First item in the list
    private int  size = 0;
}
```

1. (5 points) Within a code segment, some previous code has achieved the following circumstance: `prev` and `curr` refer to list nodes as shown in the following picture, so that `curr` immediately follows `prev` in the linked list, and a node `newNode` has been constructed.

   ![Diagram](image1.png)

   We need to change the list so that `newNode` comes between `prev` and `curr` in the list. Show **pictorially** below what needs to be done with the "next" fields to accomplish that. (You will not need to do anything with the nodes before `prev` in the list, nor anything with the nodes after `curr`.)

2. (5 points) Write the Java code segment that will accomplish **just** this task — **nothing more** (Hint: it will be very brief!)

   ```java
   newNode.next = current;
   prev.next = newNode;
   ```

3. (10 points) Write the method to add an element at the **beginning** of the list. It must correctly handle adding to an empty list and adding to a non-empty list. (Hint: this is within the class `SinglyLinkedList`, so you have direct access to `head`.) Note that you will need to construct a `Node` object. (Reminder: this linked list maintains a field with the size of the list.)

   ```java
   public void addFirst( Object item ) { // the rest is up to you
       Node newNode = new Node (item);
       newNode.next = head;
       head = newNode;
       size++; }
   ```

4. (10 points) Write the method to add an element at the **end** of the list. [Note that this implementation does not maintain a tail reference.] It must correctly handle adding to an empty list and adding to a non-empty list. (Hint: “Note” and “Reminder” as above.)

   ```java
   public void addLast ( Object item ) { // the rest is up to you
       Node newNode = new Node (item), curr = head;
       if ( head == null )
           head = newNode;
       else
           { while ( curr.next != null )
               curr = curr.next;
             curr.next = newNode;
           }
       size++; }
   ```

5. (10 points) Write the method to remove an element from the list based on its data field, return a boolean value indicating whether the removal was successful. It must correctly handle attempting to remove from an empty list and removing from the front, middle, and back of a non-empty list. Note that the specified signature passes a bare `Object` as its argument, so you must use object equality [the method “equals( Object o )” returns a boolean value] to check against a node’s data field. (Hint: Update size only when appropriate.)

   ```java
   public boolean remove( Object item ) { // the rest is up to you
       Node curr = head, prev = null;
       for ( ; curr != null && ! item.equals(curr.data);
           curr = prev.next )       prev = curr;
       if (curr == null) return false;
       if (prev == null)       head = head.next;
       else       prev.next = curr.next;
       size--;    return true; }
   ```

6. (10 points) Determine (in a “Big-O” sense) the order of the following method within `SinglyLinkedList` as a function of the number of entries in the list (in other words, think of `n` as `this.size`). Be sure to give the basis for your decision. (Hint: you may refer to algorithms discussed in class.)

   ```java
   public void six () { // the rest is up to you
       Node curr;
       for ( int nd = this.size-1; nd >= 0; nd-- )
           { curr = head;
               for ( int j = 0; j < nd; j++ )
                   { curr = curr.next;
                     }
               System.out.println ( curr.data );
           }
   }
   ```

   Inner loop goes as `n-1`, then `n-2`, then `n-3`, down to 1; in other words, we just have to add together all those numbers, so that it is exactly like selection sort. The exact quantity is `n(n-1)/2`, but the question asks for Big-O, so it is `O(n^2)`. 

**Answer Key**
7. (10 points) Determine (in a "Big-O" sense) the order of the following method as a function of the parameter \( n \). Be sure to give the basis for your decision. Hint: you may refer to algorithms discussed in class.

```java
public static void seven(int n)
{
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
            for(int k = j; k < n; k++)
                System.out.println("i: " + i + " j: " + j + " k: " + k);
}
```

The loop on \( i \) is cleanly \( n \) times. The loop on \( k \) depends on the loop on \( j \). The first time it goes \( n \) times, then \( n-1 \) times, \( n-2 \) times, etc.; in other words, we just have to add together all those numbers, so that it is nearly like the selection sort. The exact quantity is \( n(n+1)/2 \), and the outer loop make us do that \( n \) times. In terms of Big-O, that comes down to \( O(n^3) \).

8. (10 points) Determine (in a "Big-O" sense) the order of the following method as a function of the parameter \( n \). Be sure to give the basis for your decision. Hint: you may refer to algorithms discussed in class.

```java
public static void eight(int n)
{
    if(n > 0)
    {  eight (n - 1);       System.out.println("n: " + n);
        eight (n - 1);    }
}
```

This is exactly the pattern we saw in class for the Towers of Hanoi — two self-recursive calls that only decrease the problem size by one. That means it is \( O(2^n) \).

9. (10 points) Within the class `SinglyLinkedList` specified on the first page of this exam, write a recursive method that will print to the screen the entries in the linked list in reverse order. Note that you can only traverse the list in the forward direction.

```java
public void printReverse () {  printBackwards(head);  }

private void printBackwards (Node current) // the rest is up to you
{
    if ( current != null )
    {  printReverse ( current.next );
        System.out.println ( current.data );
    }
}
```

10. (10 points) You are being given a "mystery recurrence" — just the structure of the recurrence without any information about its meaning. The recurrence (and hence the recursive function) has one parameter. You are given the base case in terms of that parameter and then the recurrence (i.e., the definition of the function in terms of itself for a simpler case).

\[
\begin{align*}
\text{for } n < 1, & \text{ mystery}(n) = 0 \\
\text{for } n \geq 1, & \text{ mystery}(n) = \text{ mystery}(n-1) + 2*n - 1
\end{align*}
\]

Write the recursive Java method that reflects this recurrence and correctly computes the value. (You will not require all the space provided; it is there because the next question needs lots of room and landscape orientation for the page.)

```java
public int mystery (int n) // the rest is up to you
{  return n < 1 ? 0 : mystery(n-1) + 2*n - 1;  }
```