Name:_____

15-111/Kesden Spring 2003
Exam 2 (Retake)

Singly Linked Lists

1. Given the provided, minimal *LinkedList*, including the *Node*, please implement the method described below:

```
  /**
   * Removes the first item equal to the keyItem from  the list.
   * This method removes at most one item. If there are multiple
matching
   * items, it removes only the matching item closest to the head.
   * The list is not changed in any other way.
   * <p>
   * In the event of an error, it does not change the list,
instead it
   * returns leaving the list in its prior condition.
   * <p>
   * @param  keyItem  The first item within the list equal to this
item is
   *                  removed from the list.
   */
   public void removeFirstMatchingItem (Comparable keyItem) {

    return;
   }
```

2. Given the provided, minimal *LinkedList* class, including the *Node*, please implement the method described below:

```
/**
   * Creates and returns a new list, which contains exactly those
   * items that are present in exactly one of the two lists, but
   * not both.
   * <p>
   * It does not change either of the original lists
   * <p>
   * In the event of an error, it returns an empty list.
   * <p>
   * @param  otherList is the list that should be compared with
   * this list
   * <p>
   * @return  a new list, which contains exactly those items
   * that are present in exactly one of the two lists, but not
   * both.
   */
  public LinkedList Xor(EnhancedLinkedList otherList) {

    return null; // Remove this!
  }
```

3. Please draw a figure, or a collection of figures, that shows the evaluation of the following expression using a single stack. The figure(s) should depict the stack after each operation and should also clearly indicate the operation.

   4,5,+,3,/,2,*

4. Please draw a figure representing the call stack after each method call and method return of *someFunction (4)*, where *someFunction()* is defined as below:

```
int someFunction(int var){

  if (0 == var) return 1;
  if (1 == var) return 1;

  return someFunction (var-1) + someFunction (var-2);
}
```

5. *Quick sort* is said to have an average case runtime on the order of "nlog n". Please explain why this is the case, yet *quick sort* is $O(n^2)$

6. Is *selection sort* ever a better choice than *quick sort*? Why or why not? If so, please give an example.
7. Which sort is more efficient, *selection sort* or *bubble sort*? Why?

```
public class LinkedList
{

        public class IndexException extends Exception
        {

                public String toString()
                {
                        return ("Bad index in Linked List");
                }
        }


        private Node head;
        private Node tail;



        public LinkedList()
        {
                head = tail = null;
        }


        public void addHead(Comparable data)
        {

                Node newNode;
                newNode = new Node(data);

                newNode.setNext(head);

                head = newNode;

                if (null == tail)
                {
                        tail = head = index = newNode;
                }
        }



}
```

```
// For question #1 and question #2
public class Node
{
        private Comparable data;
        private Node next;



        public Node (Comparable data, Node next)
        {
                this.data = data;
                this.next = next;
        }


        public Node (Comparable data)
        {
                this.data = data;
                this.next = null;
        }

        public Comparable getData()
        {
                return data;
        }


        public Node getNext()
        {
                return next;
        }



        public void setNext(Node next)
        {
                this.next = next;
        }



        public void setData(Comparable data)
        {
                this.data = data;
        }
}
```

```
// For question #3
public class DNode
{
        private Comparable data;
        private DNode next;



        public DNode (Dnode prev, Comparable data, DNode next)
        {
                this.prev = prev;
                this.data = data;
                this.next = next;
        }


        public DNode (Comparable data)
        {
                this.prev = null;
                this.data =  data;
                this.next = null;
        }

        public Comparable getData()
        {
                return data;
        }


        public DNode getNext()
        {
                return next;
        }


        public DNode getPrev()
        {
                return prev;
        }



        public void setNext(DNode next)
        {
                this.next = next;
        }



        public void setPrev(DNode prev)
        {
                this.next = prev;
        }




        public void setData(Comparable data)
        {
                this.data = data;
```

```
}
```