COSC 2317.01 EXAM #01 (Fall 2016) NAME:

Question#	Points / MAX
1	/ 10
2	/ 10
3	/ 10
4	/ 10
5	/ 10
6	/ 10
7	/ 10
8	/ 10
9	/ 14
10	/ 18
TOTAL	/ 112

If needed, refer to the following UML diagrams (NOTE: in ListArrayBased, however we don't we directly manipulate array index to access proper position in the array):

Node
-item: Object -next: Node
+Node(newItem: Object) +Node(newItem: Object, nextNode: Node) +setItem(newItem: Object): void +getItem(): Object +setNext(nextNode: Node): void +getNext(): Node

ListArrayBased
-MAX_LIST: int -items: Object[] -numItems: int
+ListArrayBased() +isEmpty(): boolean +size(): int +removeAll(): void +add(index: int, item: Object): void +get(index: int): Object +remove(index: int): void

ListReferenceBased
-head: Node -numItems: int
+ListReferenceBased() +isEmpty(): boolean +size(): int +removeAll(): void +add(index: int, item: Object): void +get(index: int): Object +remove(index: int): void

1. **(10 points)** What is the final value in count after finishing the iteration? Assume count=0 before the iteration starts.

(1.1)
```
for (int i = 10; i <=  111; i++)
        count++;
```
➔ *(Your Answer)*

(1.2)
```
for (int i = 100; i >= 0; i = i - 1)
        count++;
```
➔ *(Your Answer)*

(1.3)
```
for (int i = 1; i <= 10; i++)
        for (int j = 1; j <= i; j++)
                count++;
```
➔ *(Your Answer)*

(1.4)
```
for (int i = 1; i <= 10; i++)
        for (int j = i; j >= 1; j--)
                count++;
```
➔ *(Your Answer)*

(1.5)
```
for (int i = 0; i < 10; i++){
        count = 0;
        for (int j = 1; j <=50; j  ++)
                count++;
}
```
➔ *(Your Answer)*

2. **(10 points)** Order the following growth rates in the increasing order:
$O(n)$, $O(1)$, $O(2^n)$, $O(n^3)$, $O(\log_2 n)$, $O(n^2)$, $O(n \log_2 n)$
(Your Answer)

3. **(10 points)** Compare advantages and disadvantages of implementing ADT using either an array or a reference. Fill in the blanks.

	Array-based	**Reference-based**
Size		
Access i-th item		
Overhead		

4. **(10 points)** Fill in the blanks so that the following method can **recursively do binary search** .

```
public static int binarySearch(int []array, int first, int last, int value)
{
     if (_____________________________________________)
          return -1;
     int mid = _____________________________________________;
     if (value < array[mid])
               return  _____________________________________________;
     else if (value == array[mid])
               return  _____________________________________________;
     else
               return _____________________________________________;
}
```

5. **(10 points)** Fill in the blanks so that the following method can **non-recursively do binary search**.

```
public static int binarySearch(int []array, int value)
{
     int first = 0;
     int last = _____________________________________;
     while (first <= last) {
          int mid = _____________________________________________;
          if (value < array[mid])
               _____________________________________________;
          else if (array[mid] == value)
               _____________________________________________;
          else
               _____________________________________________;
     }
     return -1;
}
```

6. **(10 points)** The following method (***reverse()***) is not correctly implemented. Correct only the errouneous statements so that the mathods can iteratively reverse a singly linked list. Assume that the method header is correct. (Hint: There are five logical/syntax errors.)

(Your Correction)

```
public void reverse()
{
     Node prev = head;                    ➔ _____
     Node curr = head;                    ➔ _____
     while (curr == null){                ➔ _____
          Node next = curr.getNext();     ➔ _____
          curr.getNext(prev);             ➔ _____
          curr = prev;                    ➔ _____
          curr = next;                    ➔ _____
     }
     curr = prev;                         ➔ _____
  }
```

7. **(10 points)** Fill in the blanks so that the following method can recursively reverse a singly linked list. Note that you also need to implement **reverseRecursive()**, which is a helper method that is called in a driver class like aList.reverseRecursive().

```
public Node reverse(Node prev, Node curr)
{
     if (curr == null)
          return _____________________________________________;
     Node next = curr.getNext();
     ____________________________________________________________;

     return _____________________________________________________;
}

 public void reverseRecursive()
{
     _________________ = ________________________________________;
}
```

8. **(10 points)** Correct the incorrect code for the **Node** class that we have learned. (Hint: There are five logical/syntactic errors)

```
public class Node {

     private Object item;
     private Object next;

     public Node(Object newItem) {
          item = newItem;
          next = null;
     } // end constructor

     public Node(Object newItem, Node nextNode) {
          newItem = item;
          nextNode = next;
     } // end constructor

     public void setItem(Object newItem) {
          item = newItem;
     } // end setItem

     public Object getItem() {
          return item;
     } // end getItem

     public void setNext(Node nextNode) {
          nextNode = next;
     } // end setNext

     public void getNext() {
          return next;
     } // end getNext

} // end class Node
```

9. **(1π points)** Implement ListA arrayBased Class

```
public class ListArrayBased implements ListInterface {

      private static final int MAX_LIST = 50;
      private Object items[];  // an array of list items
      private int numItems;  // number of items in list

      public Object get(int index) throws ListIndexOutOfBoundsException {
            if (index >= 0 && index < numItems)

                  return items[(9.1)_____________________________________________];
            else
                  throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException");
      } // end get


      public void add(int index, Object item) throws ListIndexOutOfBoundsException {
            if (numItems >= MAX_LIST)
                  throw new ListException("ListException on add");
            if (index >= 0 && index <= numItems) {
                  for (int pos = numItems-1; pos >= index; pos--)

                        items[pos+1] = items[(9.2)_____________________________________];

                  items[index] = (9.3)_____________________________________________;

                  (9.4)_____________________________________________________________;
            } else
                  throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException”);
      } //end add


      public void remove(int index) throws ListIndexOutOfBoundsException {
            if (index >= 0 && index < numItems){

                  for (int pos = index+1; pos < (9.5)_____________________________; pos++)

                        items[pos-1] = items[(9.6)_____________________________________];

                  (9.7)_____________________________________________________________;
            } else
                  throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException");
      } // end remove


}  // end ListArrayBased
```

10. **(18 points)** Implment ListR eferenceBased class.

```
public class ListReferenceBased implements ListInterface {
      private Node head;
      private int numItems; // number of items in list

      public Object get(int index) throws ListIndexOutOfBoundsException {
            if (index >= 0 && index <= numItems) {
                  Node curr = head;
                  for (int skip = 0; skip < index; skip++)
                        curr = curr.getNext();

                  return (10.1)_____;
            } else
                  throw new ListIndexOutOfBoundsException("index out of bounds on get")
      } // end get

      public void add(int index, Object item) throws ListIndexOutOfBoundsException {
            if (index >= 0 && index <= numItems) {

                  if (index == (10.2)_____)

                        head = new Node(item, (10.3)_____);
                  else {
                        Node prev = head;
                        for (int skip = 0; skip < index-1; skip++)
                              prev = prev.getNext();
                        Node newNode = new Node(item, (10.4)_____);
                        prev.setNext(newNode);
                  } // end if

                  (10.5)_____;
            } else
                  throw new ListIndexOutOfBoundsException(“index out of bounds on add");
      }  // end add

      public void remove(int index) throws ListIndexOutOfBoundsException {
            if (index >= 0 && index < numItems) {

                  if (index == (10.6)_____)

                        head = (10.7)_____;
                  else {
                        Node prev = head;
                        for (int skip = 0; skip < index-1; skip++)
                              prev = prev.getNext();
                        Node curr = prev.getNext();

                        prev.setNext((10.8)_____);
                  } // end if

                  (10.9)_____;
            } else
                  throw new ListIndexOutOfBoundsException(“index out of bounds");
      }   // end remove

 } // end ListReferenceBased
```