EXAM #01 NAME:

Question#	Points / MAX
1	/ 10
2	/ 5
3	/ 5
4	/ 10
5	/ 10
6	/ 10
7	/ 10
8	/ 10
9	/ 10
10	/ 20
TOTAL	/ 100

NOTE that for all the questions that implement methods for singly linked list, **<u>you are required to call only the methods of the classes in chapters 4 & 5</u>** (refer to UML diagrams given below):

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

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 = 1; i <= 100; i++)
        count++;
```
➔ *(Your Answer)*

(1.2)
```
for (int i = -10; i < 150; i++)
        count++;
```
➔ *(Your Answer)*

(1.3)
```
for (int i = 0; i <= 10; i++)
        for (int j = 0; j <=20; j++)
                count++;
```
➔ *(Your Answer)*

(1.4)
```
for (int i = 0; i < 10; i++)
        for (int j = 0; j <20; j++)
                count++;
```
➔ *(Your Answer)*

(1.5)
```
for (int i = 0; i < 10; i++)
        for (int j = 0; j <=20; j++)
                count++;
```
➔ *(Your Answer)*

2. **(5 points)** Order the following growth rates in the increasing order:

$O(1)$, $O(2^n)$, $O(n^2)$, $O(n^3)$, $O(n \log_2 n)$, $O(\log_2 n)$, $O(n)$

(Your Answer)

3. **(5 points)** Compare advantages and disadvantages of implementing ADT using either an array or a reference. Choose a word or write your answer after question mark(?).

	Array-based	**Reference-based**
Size	[✓ fixed] / not-fixed	fixed / [✓ not-fixed]
Access i-th item	[✓ direct] / indirect	direct / [✓ indirect]
Overhead	? **Very Small (Big 0(1))**	? **Very High (Big O(N))**

4. **(10 points)** Write **recursive binary search** (say **binarySearch()**)method in Java.

(Your Answer)

```
public static int binarySearch(int[] array, int first, int last, int value)
{
                        if (first > last){
                          return -1;
                              }
                      int mid = first + last / 2
                      if (value < array[mid]){
          return binarysearch(int[] array, int first, int mid, int value);
                              }
                    else if (value == array[mid]){
                        return array[mid];
                              }
                          else return
          return binarysearch(int[] array, int mid, int last, int value);
                              }
```

5. **(10 points)** Write **non-recursive binary search** (say ***binarySearch()***) method in Java.

(Your Answer)

```
public static int binarySearch(int[] array, int value)
{
```

6. **(10 points)** Write a method (say ***reverse()***) that iteratively reverse a singly linked list.

(Your Answer)

```
public void reverse()
{
```

7. **(10 points)** Write a method (say ***reverse(Node prev, Node curr)***) that 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().

(Your Answer)

```
public Node reverse(Node prev, Node curr)
{
```

```
 public void reverseRecursive()
 {
```

8. **(10 points)** Given a singly linked list, devise a time- and space-efficient method in java (say **findMToLast()**) to find the *m*th-to-last element of the list. Define mth to last such that when m=0, the last element of the list is returned.

(Your Answer)

```
public Node findMToLast(int m)
{
```

9. **(10 points)** Convert the following infix expressions to either prefix or postfix form as requested.

(9.1) A – B + C ➔ (prefix form)

(9.2) A / (B * C) ➔ (prefix form)

(9.3) A + B * C ➔ (postfix form)

(9.4) (A + B) * C ➔ (postfix form)

(9.5) A + B * C - D ➔ (postfix form)

10. **(20 points)** Consider the following recursive method:

```
public static int method(int x, int n)
{
     if (n == 0)  return   1;
     else         return   x  *  method(x,  n - 1);
}
```

(10.1) Write a recursive definition of method(x, n).
(Your Answer)

$$\text{method(x, n)} = \begin{cases} \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ & if \quad \_\_\_\_\_\_\_\_ \\ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ & if \quad \_\_\_\_\_\_\_\_ \end{cases}$$

(10.2) What does this method do?

(10.3) Prove that your answer to (10.1) & (10.2) is correct by using mathematical induction.
(Your Answer)

EXAM #01 (make-up) NAME:

Question#	Points / MAX
1	/ 10
2	/ 10
3	/ 10
4	/ 10
5	/ 10
6	/ 10
7	/ 10
8	/ 10
9	/ 10
10	/ 10
TOTAL	/ 100

If needed, refer to the following UML diagrams (NOTE: the textbook define the two methods, **translate()** and **find()**, in ListArrayBased, however we don't use the method in this test. Instead, 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 = 12; 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 <=20; 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){             →      while (curr != null){
        Node next = curr.getNext();   → ______________________________
        curr.getNext(prev);           →    curr.setNext(previous) ???
        curr = prev;                  →         prev = curr
        curr = next;                  → ______________________________
    }
    curr = prev;                      →     head = 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. **(10 points)** Answer the questions on ListArrayBased class (and ListReferenceBased class).

(9.1) (TRUE or FALSE) *The implementation (i.e., body) of ListArrayBased() is identical to that of removeAll().*

(9.2) (TRUE or FALSE) *The implementation (i.e., body) of isEmpty() in ListArrayBased class is identical to that of isEmpty() in ListReferenceBased class.*

(9.3) (TRUE or FALSE) *The implementation (i.e., body) of size() in ListArrayBased class is identical to that of size() in ListReferenceBased 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[(9.4)____________________________________________];
            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[(9.5)________________________________];

                  items[index] = (9.6)________________________________________;

                  (9.7)________________________________________________________;
            } 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 < (9.8)__________________________; pos++)

                        items[pos-1] = items[(9.9)________________________________];

                  (9.10)_______________________________________________________;
            } else
                  throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException");
      } // end remove


}  // end ListArrayBased
```

10. **(10 points)** Answer the questions on ListReferenceBased class.

(10.1) *(TRUE or FALSE) The implementation (i.e., body) of ListReferenceBased() is identical to that of removeAll()*

```
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 (10.2)____________________________________________________;
            } 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 == (10.3)______________________________________________)

                        head = new Node(item, (10.4)_______________________________);
                  else {
                        Node prev = head;
                        for (int skip = 0; skip < index-1; skip++)
                              prev = prev.getNext();
                        Node newNode = new Node(item, (10.5)_______________________);
                        prev.setNext(newNode);
                  } // end if

                  (10.6)___________________________________________________________;
            } 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 == (10.7)______________________________________________)

                        head = (10.8)______________________________________________;
                  else {
                        Node prev = head;
                        for (int skip = 0; skip < index-1; skip++)
                              prev = prev.getNext();
                        Node curr = prev.getNext();

                        prev.setNext((10.9)________________________________________);
                  } // end if

                  (10.10)__________________________________________________________;
            } else
                  throw new ListIndexOutOfBoundsException("index out of bounds");
      }   // end remove

 } // end ListReferenceBased
```

EXAM #01 NAME:

Question#	Points / MAX
1	/ 10
2	/ 5
3	/ 5
4	/ 10
5	/ 10
6	/ 10
7	/ 10
8	/ 10
9	/ 10
10	/ 10
11	/ 10
TOTAL	(+ 5) / 100

If needed, refer to the following UML diagrams (NOTE: the textbook define the two methods, **translate()** and **find()**, in ListArrayBased, however we don't use the method in this test. Instead, we directly manipulate array index to access proper position in the array. Also note that index starts from 0, not from 1.):

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 = 1; i < 100; i++)
        count++;
```
➔ *(Your Answer)*

(1.2)
```
for (int i = -10; i <= 150; i++)
        count++;
```
➔ *(Your Answer)*

(1.3)
```
for (int i = 0; i < 10; i++)
        for (int j = 0; j < 20; j++)
                count++;
```
➔ *(Your Answer)*

(1.4)
```
for (int i = 0; i <= 10; i++)
        for (int j = 0; j <20; j++)
                count++;
```
➔ *(Your Answer)*

(1.5)
```
for (int i = 0; i < 10; i++)
        for (int j = 0; j <=20; j++)
                count++;
```
➔ *(Your Answer)*

2. **(5 points)** Order the following growth rates in the increasing order:

$O(n)$, $O(n^3)$, $O(2^n)$, $O(n^2)$, $O(\log_2 n)$, $O(n\log_2 n)$, $O(1)$

(Your Answer)

3. **(5 points)** Compare advantages and disadvantages of implementing ADT using either an array or a reference. Write your answer after each question mark(?).

	Array-based	**Reference-based**
Size	?	?
Access item	?	?
Overhead	?	?

4. **(10 points)** Write **recursive binary search** (say **binarySearch()**)method in Java.

(Your Answer)

```
public static int binarySearch(int[] array, int first, int last, int value)
{



}
```

5. **(10 points)** Write **non-recursive binary search** (say ***binarySearch()***) method in Java.

(Your Answer)

```
public static int binarySearch(int[] array, int value)
{



}
```

6. **(10 points)** Draw diagrams that describe how the following JAVA objects refer memory.

(6.1) ***Integer i, j;***	
(6.2) ***i = new Integer(10);***	
(6.3) ***j = new Integer(20);***	
(6.4) ***j = i;***	
(6.5) ***i = null;***	

(EXTRA 5 POINTS) *Define the following terms (in Java)*

(1) ***Overloading:***

(2) ***Overriding:***

7. **(10 points)** Fill in the blank and also correct the code for the **ListInterface** (Hint: Refer to the UML on the first page.)

```
public ____________________ ListInterface {

      // list operations:

      public Booldean isEmpty();

      public int size();

      public Object add(int index, Object item)
                  throws ListIndexOutOfBoundsException;
      public Object remove(int index)
                  throws ListIndexOutOfBoundsException;
      public Object get (int index)
                  throws ListIndexOutOfBoundsException;
      public Object removeAll();

} // end of ListInterface
```

8. **(10 points) TRUE** or **FALSE** (Refer to the UMLs and Questions 10 & 11.)

(8.1) (**TRUE** or **FALSE**) The implementation (i.e., body) of ListArrayBased() is identical to that of removeAll().

(8.2) (**TRUE** or **FALSE**) The implementation (i.e., body) of isEmpty() in ListArrayBased class is identical to that of isEmpty() in ListReferenceBased class.

(8.3) (**TRUE** or **FALSE**) The implementation (i.e., body) of size() in ListArrayBased class is identical to that of size() in ListReferenceBased class.

(8.4) (**TRUE** or **FALSE**) The implementation (i.e., body) of ListReferenceBased() is identical to that of removeAll()

(8.5) (**TRUE** or **FALSE**) Both ListArrayBased and ListReferenceBased classes implement ListInterface.

9. **(10 points)** Correct the incorrect code for the **Node** class that we have learned. (Hint: Refer to the UML on the first page.)

```
public class Node {

    private Node 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
```

10. **(10 points)** Answer the questions on ListArrayBased class (NOTE that as we discussed in the class, every index starts from 0).

```
public class ListArrayBased (10.1)_________________________ ListInterface {

    private static (10.2)______________________________ 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 && (10.3)_____________________________________________)

            return items[(10.4)___________________________________________];
        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[(10.5)______________________________];

            items[index] = (10.6)_______________________________________;

            (10.7)______________________________________________________;
        } 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 < (10.8)________________________; pos++)

                items[pos-1] = items[(10.9)____________________________];

            (10.10)_____________________________________________________;
        } else
            throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException");
    } // end remove

}  // end ListArrayBased
```

11. **(10 points)** Answer the questions on ListReferenceBased class (NOTE that as we discussed in the class, every index starts from 0.).

```
public class ListReferenceBased (11.1)____________________________ 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 (11.2)__________________________________________________;
                } 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 == (11.3)____________________________________________)

                                head = new Node(item, (11.4)______________________________);
                        else {
                                Node prev = head;
                                for (int skip = 0; skip < index-1; skip++)
                                        prev = prev.getNext();
                        Node newNode = new Node(item, (11.5)_______________________);
                                prev.setNext(newNode);
                        } // end if

                        (11.6)________________________________________________________;
                } 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 == (11.7)____________________________________________)

                                head = (11.8)____________________________________________;
                        else {
                                Node prev = head;
                                for (int skip = 0; skip < index-1; skip++)
                                        prev = prev.getNext();
                                Node curr = prev.getNext();

                                prev.setNext((11.9)______________________________________);
                        } // end if
                        (11.10)_______________________________________________________;
                } else
                        throw new ListIndexOutOfBoundsException("index out of bounds");
        }   // end remove
} // end ListReferenceBased
```

10. (10 points) Answer the questions on ListArrayBased class (NOTE that as we discussed in the class, every index starts from 0).

```
public class ListArrayBased _________________________ ListInterface {

    private static _______________________________ 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[__________________________________________________];
        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 __________________ && index __________________________________) {

            for (int pos = numItems - 1; pos ________________________________; pos--)

                items[__________________] = items[pos];

            items[index] = ___________________________________________________;

            numItems++;
        } 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 __________________________________; pos++)

                items[________________] = items[pos];

            numItems--;
        } else
            throw new ListIndexOutOfBoundsException("IndexOutOfBoundsException");
    } // end remove

}  // end ListArrayBased
```

11. (10 points) Answer the questions on ListReferenceBased class (NOTE that as we discussed in the class, every index starts from 0.).

```
public class ListReferenceBased ____________________________ 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 =___________________________________________________;
                        for (int skip = 0; skip < index; skip++)

                                curr = ____________________________________________________;
                        return  curr.getItem();
                } 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 == 0 )
                                head = new Node(item, _______________________________________);
                        else {
                                Node prev = __________________________________________________;
                                for (int skip = 0; skip < index-1; skip++)

                                        prev = ______________________________________________;
                        Node newNode = new Node(item,  prev.getNext());
                                prev.setNext(newNode);
                        } // end if
                        numItems++;
                } 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 == 0)

                                head = _______________________________________________________;
                        else {
                                Node prev = ___________________________________________________;
                                for (int skip = 0; skip < index-1; skip++)

                                        prev = _______________________________________________;
                                Node curr = prev.getNext();

                                prev.setNext(____________________________________________);
                        } // end if
                        numItems--;
                } else
                        throw new ListIndexOutOfBoundsException("index out of bounds");
        }   // end remove
 } // end ListReferenceBased
```