

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, **you are required to call only the methods of the classes in chapters 4 & 5** (refer to UML diagrams given below):

Node	ListReferenceBased
-item: Object -next: Node	-head: Node -numItems: int
+Node(newItem: Object) +Node(newItem: Object, nextNode: Node) +setItem(newItem: Object): void +getItem(): Object +setNext(nextNode: Node): void +getNext(): Node	+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 $O(1)$)	? 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 *m*th 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) $A - B + C$ \rightarrow (prefix form)

(9.2) $A / (B * C)$ \rightarrow (prefix form)

(9.3) $A + B * C$ \rightarrow (postfix form)

(9.4) $(A + B) * C$ \rightarrow (postfix form)

(9.5) $A + B * C - D$ \rightarrow (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 $\text{method}(x, n)$.

(Your Answer)

$$\text{method}(x, n) = \begin{cases} \underline{\hspace{2cm}} & \text{if } \underline{\hspace{2cm}} \\ \underline{\hspace{2cm}} & \text{if } \underline{\hspace{2cm}} \end{cases}$$

(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++) ➔ (Your Answer)
 count++;

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

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

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

(1.5) for (int i = 0; i < 10; i++){ ➔ (Your Answer)
 count = 0;
 for (int j = 1; j <= 20; j++)
 count++;
 }

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 erroneous statements so that the methods 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;
}
```

→ _____

→ _____

→ **while (curr != null){** _____

→ _____

→ **curr.setNext(previous) ???** _____

→ **prev = curr** _____

→ _____

→ **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) <i>Integer i, j;</i>	
(6.2) <i>i = new Integer(10);</i>	
(6.3) <i>j = new Integer(20);</i>	
(6.4) <i>j = i;</i>	
(6.5) <i>i = null;</i>	

(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 Boolean 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
```