1. Complexity of methods is usually measured only insofar as their time and space resource requirements are concerned. Of these two, which one do you judge to be more important when deciding which of alternative algorithms to implement in order to solve a problem (i.e., given two algorithm choices where one saves in time requirements while the other saves in space requirements, which would you opt for)? How often do you think you can find a solution that actually improves on both time and space resource requirements? (10 pts.)

Time seems to be the more important consideration of the two. One can always acquire more memory, but once time is lost, it cannot ever be recovered. Time resource here does not mean just the running time. Very often, the human time resource is even more important in terms of cost. Saving on development time may mean more significant dollar savings than saving on run time.

Being able to come up with a solution that saves on both time and space is a big challenge. We up against a virtual tautology here—the time-space tradeoff. What we usually gain in time, we must give back in terms of more space usage. What we can save in space usually means we have to spend more time being more clever with how we manipulate the data.
2. Discuss substantially the advantages and disadvantages of using BoundedList versus using LinkedList as implementations of the abstract List class. Is DynamicList a happy middle ground? Explain your answer. (10 pts.)

The use of BoundedList gives us the ease and efficiency of indexed array access. This comes at a price (it usually does - see No. 1 above): we allocate array space even if we do not actually use that space.

The use of LinkedList gives us the comfort of knowing that we allocate space only when we do need it. This comes at a price (it usually does - see no. 1 above): indexed access is now expensive, needing usually O(n) whereas it was O(1) for boundedlist.

Whether DynamicList is a happy middle ground can be debated. We allocate an initial capacity, which means that space is still unused (at the start). But when we do need to expand the capacity, the vector-based implementation is able to accommodate the adjustment. Indexed access is still O(1).

All in all, it becomes more and more convincing that DynamicList is a happy compromise.
Part II. **MULTIPLE CHOICE.** Select from among the given options the best answer for the given question. Record the answers in the supplied scantron sheet. *(1 pt. each)*

1. The ________ of a program is a measure of its efficiency and is usually expressed using the “big-O” notation.
   - a. syntax
   - b. semantics
   - c. preconditions
   - d. postconditions
   - e. computational complexity

2. ______ reflects the intrinsic difficulty of solving it and forms a lower bound to the complexity of all solutions.
   - a. Problem analysis
   - b. Algorithm analysis
   - c. Problem symmetry
   - d. Problem complexity
   - e. Complexity analysis

3. An ______ problem is one that admits only solutions that are exponential in time complexity.
   - a. unsolvable
   - b. interesting
   - c. intractable
   - d. inoperable
   - e. incapable

4. When a method’s complexity matches the problem’s, we say that the method is ______.
   - a. even
   - b. optimal
   - c. complete
   - d. well-behaved
   - e. exponential

5. A(n) ______ problem is one which admits no known solutions (e.g., the “halting problem” from computability theory).
   - a. unsolvable
   - b. exponential
   - c. intractable
   - d. philosophical
   - e. pathological

6. We assess a(n) ______ a cost of $O(1)$.
   - a. elementary statement
   - b. union statement
   - c. iteration statement
   - d. compound statement
   - e. composite statement

7. _____ forms an upper bound on the resource required by all possible problem instances.
   - a. Worst-case complexity
   - b. Best-case complexity
   - c. Average-case complexity
   - d. Universal complexity
   - e. None of these answers

8. _____ is the measure of the resource required when amortized over all possible cases.
   - a. Worst-case complexity
   - b. Best-case complexity
   - c. Average-case complexity
   - d. Universal complexity
   - e. None of these answers
9. _____ forms a lower bound on the resource required by all possible problem instances.
   a. Worst-case complexity
   b. Best-case complexity
   c. Average-case complexity
   d. Universal complexity
   e. None of these answers

10. The process of determining an algorithm’s properties (e.g., correctness, complexity, etc.) is called _______.
    a. algorithm analysis
    b. algorithm implementation
    c. case analysis
    d. algorithm calculus
    e. All of these answers are valid.

11. This class implements lists using a fixed size array as the main design aspect.
    a. CourseList class
    b. StudentList class
    c. LinkedList class
    d. DynamicList class
    e. BoundedList class

12. In an array, these array elements are required to be homogeneous:
    a. array components
    b. array size
    c. array limit
    d. array memory
    e. All of these are valid answers.

13. When this is encountered in an array operation, then the run-time exception ArrayIndexOutOfBoundsException is thrown:
    a. index is initialized to 0
    b. index used is negative in value
    c. index used is less than array size
    d. index used is equal to size
    e. Both (b) and (d) are valid answers.

14. An exact duplicate of another object is what this term refers to:
    a. copy
    b. clone
    c. facsimile
    d. analog
    e. substitute

15. For a class to properly contain a clone() method, it has to implement this “tag” interface:
    a. Serializable
    b. Copyable
    c. Order
    d. Cloneable
    e. Sortable

16. Usually defined as a protected feature, this is invoked by concrete subclasses in creating its instances.
    a. instance variables
    b. asphalt constructor
    c. mixer constructor
    d. default constructor
    e. abstract constructor

17. Used by the abstract class DynamicList to improve on BoundedList’s fixed-size implementation.
    a. Cloneable interface
    b. dynamic arrays
    c. clone() method
    d. do-while loops
    e. for loops
18. Class provided by Java that provides “dynamic” list functionality except for the requirement of homogeneity in its member elements.
   a. java.util.Vector
   b. java.lang.Array
   c. java.awt.List
   d. javax.swing.JPanel
   e. java.util.Random

19. When a class provides the needed functionality but not the correct interface, one can incorporate instances of the class inside a(n) ____ class.
   a. throw-away
   b. internal
   c. swing
   d. wrapper or adaptor
   e. None of these is a valid answer.

20. This is the main advantage of array-based implementations and is the reason why indexed list access is an \(O(1)\) proposition.
   a. direct access to array components
   b. linked access to array components
   c. creative complexity analysis
   d. concise complexity analysis
   e. equivalency of \texttt{clone()} & \texttt{copy()}

21. These are the basic components of the linked structures used in \texttt{LinkedList} and \texttt{DoublyLinkedList}.
   a. chains
   b. nodes
   c. sequences
   d. GUIs
   e. private inner classes

22. A special kind of node not meant to contain a list element but used to “stand for” an empty list and simplify implementation logic:
   a. null node
   b. logical node
   c. empty node
   d. index node
   e. header node

23. This variant of linked lists uses one of the nodes' next field to complete the circuit thus allowing for a traversal of all of the list elements starting from any of the other nodes.
   a. circular lists
   b. redundant lists
   c. arbitrary lists
   d. primitive lists
   e. None of these answers is valid.

24. This variant of linked lists makes use of nodes that have both \texttt{next} and \texttt{previous} components in addition to the element component.
   a. binary lists
   b. tertiary lists
   c. doubly-linked lists
   d. circular lists
   e. trinary lists

25. Memory allocation when an object is created (and not before) is called:
   a. creationist storage allocation
   b. evolutionary storage allocation
   c. just-in-time storage allocation
   d. dynamic storage allocation
   e. static storage allocation
26. The inner class that encapsulates node functionality in linked list implementations is ______.
   a. protected class Node
   b. private class Node
   c. public class Node
   d. class Node
   e. interface Node

27. The abstract class that encapsulates unbounded list functionality is:
   a. LinkedList
   b. BoundedList
   c. DynamicList
   d. StudentList
   e. CourseList

28. An erroneous situation arising from reclaiming storage that is still accessible is called:
   a. still-borne memory reclamation
   b. not-in-time memory management
   c. premature garbage collection
   d. dangling reference
   e. dynamic allocation

29. An erroneous situation resulting from inaccessible storage not being reclaimed.
   a. memory leak
   b. cyber-dementia
   c. dangling reference
   d. not-on-time garbage collection
   e. InappropriateReclaimError

30. This is the main advantage of linked-list implementations at the price of indexed list access being an \( O(n) \) proposition.
   a. direct access to array components
   b. linked access to array components
   c. no a priori bound on list size
   d. no memory leaks
   e. no dangling references

Study the static class diagrams below – they will be helpful in answering some of the questions on the following pages:
Part III. SHORT CODING. Supply the answers that are requested based on the description of the programming situation provided.

1. To provide a means of representing a poker hand (five cards from a deck of cards), supply the formal definition of class PokerHand which extends class CardList (just the first lines of code is needed here, not the complete code for the class). Base your answer on the static diagram on the previous page. (5 pts.)

```java
public class PokerHand extends CardList {
    public PokerHand(int capacity) {
        super(capacity);
    }
    : 
}
```

2. To provide a means of representing the students in CSCI 2125 this summer, we have defined the following class (as illustrated in the static diagram on the previous page):

```java
public class CSCIList extends StudentList {
    : 
}
```

Needing functionality for answering queries about the size of a section of CSCI 2125, the following method is specified:

```java
public int classSize(CSCIList section) {
    : 
}
```

Supply the implementation of this method: (5 pts.)

```java
public int classSize(CSCIList section) {
    return section.size();
}
```
Part IV. LESS SHORT CODING. Supply the answers that are requested.

1. Consider a modified version of class LinkedList. We give part of its implementation below:

```
/**
 * A linked implementation of lists with
 * a slight modification.
 */
public abstract class LinkedList
    implements Cloneable {

    /**
     * Create an empty LinkedList.
     */
    protected LinkedList() {
        size = 0;
        last = null;
    }
    //continued next column

    //instance variables
    private int size;
    private Node last;

    //private inner class
    private class Node {
        /**
         * Create a Node containing
         * the specified element.
         */
        public Node( Object element ) {
            this.element = element;
            this.next = null;
        }
    } //end LinkedList
```

In this variant, instead of having a reference to the first node of the Node sequence, we have one for the last node. We then use the last node’s next component to reference the first node in the Node sequence:

```
In this variant, all of the next components of all Node instances are in use and recovering the first node is as easy as using a local variable to store its location:

    Node first = this.last.next; //stores reference to first node
```

Your task is to supply the implementation of two important methods: append() and copy(). For the former, you are asked to supply the method body in its totality. In the latter, you are asked to fill in important missing elements in the code that implements the method.
(a) Firstly, complete the implementation of the method `append()` that takes an object and makes it the new last element in the current list. The implementation should reflect the variant described on the previous page. As a point of reference, we give the original implementation of the method (see bottom of this page). (10 pts.)

```java
/** *
 * Append the specified object to the end of this list *
 */
public void append( Object obj ) {
    if ( this.isEmpty() ) {
        last = new Node( obj );
        last.next = last;
    } else {
        Node newLast = new Node( obj );
        newLast.next = last.next;
        last.next = newLast;
        last = newLast;
    }
    size = size + 1;
}
```

(b) Assess the method complexity of your implementation. Give a short explanation of your answer: (5 pts.)

Since we do not have to "look for" the last element in the node sequence, the method does the append in $O(1)$ time. We have the variable `last` to thank for this.
(c) Next, supply the missing program elements in the implementation of the `copy()` (and `clone()`) commands below:

```java
public Object clone() {
    try {
        // clone the "top level" object by invoking
        // clone method of parent class (i.e., Object)
        LinkedList theCopy =
            (LinkedList) super.clone(); // top level object
        // replicate the linked structure node by node
        // by repeatedly "appending" list objects in nodes
        if (this.size > 0) { // we have elements to copy!
            Node ptr = this.last.next; // point to first node
            int ctr = 0;               // counter of nodes already copied
            theCopy.clear();           // "empty" the copy
            while (ctr < this.size) {  // (2 pts.)
                theCopy.append(ptr.element); // append element from original
                ctr = ctr + 1;               // (2 pts.)
                ptr = ptr.next;             // (2 pts.)
            }
        }
        return theCopy; // return the copy of current list
    } catch (CloneNotSupportedException e) {
        return null;
    }
}
```

```java
public LinkedList copy() {
    return (LinkedList) this.clone(); // invoke clone
} // end copy
```
(d) Finally, assess the method complexity of the clone() command on the previous page. Briefly explain your answer. (5 pts.)

The method has a very prominent loop that copies each of the list element from the “original” list to the list-clone. This makes the method complexity of clone() at $O(n)$.

BONUS: Provide the implementation of the query indexOf() which returns the index of the given object’s earliest occurrence in the list or -1 if it is not in the list. The formal specification is given below: (5 pts.)

```java
public int indexOf( Object obj ) {

    for ( int i=0; i<this.size(); i=i+1 )

        if ( this.get(i).equals( obj ) )
            return i;

    return -1;

} //end indexOf
```