1) A list in which each stored element is associated with a reference to its successor is called
   A) an array list
   B) a map
   C) a linked list
   D) None of the above
   Answer: C

2) To allocate storage for its elements, an array-based list such as `ArrayList` uses
   A) linked allocation
   B) contiguous allocation
   C) capacity allocation
   D) fixed size allocation
   Answer: B

3) To allocate storage for their elements, linked lists use
   A) random allocation
   B) linked allocation
   C) expandable allocation
   D) contiguous allocation
   Answer: B

4) A linked list is represented by a reference to
   A) the first node in the list, unless the list is empty, in which case the reference is set to null
   B) to the list representation object, which contains a boolean flag set to false when the list is empty
   C) the superclass of the list
   D) None of the above
   Answer: A

5) In Java, the first node in a list has index
   A) -1
   B) 0
   C) 1
   D) 2
   Answer: B

6) A `Node` class for a linked list that can hold elements of type `Object` can be declared to have fields
   A) `Object element;`
   B) `Object element; Node next;`
   C) `Object element; Node *next;`
   D) `Object element; next element;`
   Answer: B

7) In a linked list, the predecessor of a node X
   A) is undefined if X is the first node, otherwise it is the node whose index is one less than the index of X
   B) is the first node in the list
   C) is the node that comes after X
   D) is the node that is just before X
   Answer: A

8) The objects that form the units of memory allocation in a linked lists are called
   A) memory modules
   B) elements
   C) nodes
   D) links
   Answer: C

9) In a linked list, the successor of a node X
   A) may not exist
   B) is the node that comes after X, and it always exists
   C) is the last node in the list
   D) is the node whose index is one greater than the index of X

   Answer: A

10) To remove the first node in a nonempty linked list,
   A) move the successor reference in the head node one node forward:
   ```
   head.next = head.next.next;
   ```
   B) set a reference `pred` to the predecessor of the node you want to remove, and set the successor of `pred` to the successor of the head
   C) move the head reference one node forward:
   ```
   head = head.next;
   ```
   D) delete the node by setting the head reference to null:
   ```
   head = null;
   ```

   Answer: C

11) To remove a node X with index 1 or greater from a linked list,
   A) set a reference `pred` to the predecessor of the node you want to remove, and set the successor of `pred` to the successor of X
   B) decrease the index of the node to be removed by 1, and then set its reference to `null`
   C) first store the value of the element in X in a safe place, then set its reference to `null`
   D) None of the above

   Answer: A

12) A systematic procedure for starting at the first node in a list, and visiting all nodes in the list by going from each node to its successor is called
   A) a sweep
   B) a traversal
   C) travelling over the list
   D) All of the above

   Answer: B

13) In a linked list implementation using a reference `first` to point to the first node of the list, a method `isEmpty()` can test to see if the list is empty by executing the statement(s)
   A) `return null;`
   B) `first = null; return first;`
   C) `if (isEmpty()) return true; else return false;`
   D) `return first == null;`

   Answer: D

14) A list method void `add(int index, E x)` seeking to add an element *x* that is an object of a class *E* to a list, should throw an `IndexOutOfBoundsException when`
   A) the index is negative, or greater than the size of the list
   B) the index is negative, or greater than, or equal to, the size of the list
   C) the index is greater than or equal to the size of the list
   D) The element *x* falls outside of the bounds specified by the index

   Answer: A

15) A method `int size( )` in a linked list class returns the number of elements stored in the list by executing the single statement `return count;` For this to work correctly,
   A) `count` must be a static variable in the class, initialized to 0, incremented by each `add` method, and decremented by each `remove` method
   B) `count` must be an instance variable in the class, initialized to 0, incremented by each `add` method, and decremented by each `remove` method
   C) `count` must be a local variable in each `add` and `remove` method: the add methods must increment `count`, and the remove methods must decrement `count`.
   D) It is not possible for `size` to just return the count value: the `size` method must set `count` to 0 and then increment `count` once for each element in the list.

   Answer: B

16) A linked list class keeps its elements in the order in which they are added, with the index of an element X being greater than the index of any element added to the list before X. Addition of new elements to such a list can be made more efficient
   A) by keeping a reference to the element with the least index
   B) by starting at the head, and quickly traversing the list to locate the place where a new element should be added
   C) by keeping a reference to the last element added
   D) None of the above

   Answer: C

17) A list method `E remove(int index)` designed to remove and return the element at the given index should throw `IndexOutOfBoundsException` when
   A) the index is 0
   B) the index is negative, and greater than the size of the list
   C) the index is negative, or is greater than the size of the list
   D) the index is negative, or is greater than, or equal to, the size of the list

   Answer: D

18) To remove a node with a positive index *k* from a linked list,
   A) decrement *k* by 1, and set the reference to the node to be removed to null
   B) start a reference *r* at the head of the list, walk *r* forward *k* steps, and then set *r* to null
   C) assign the successor reference in the node with index *k* to the successor reference in the node with index *k*-1
   D) decrement *k* by 1, and then use recursion

   Answer: C

19) To remove a node with index 0 from a linked list,
   A) just move the head reference one node forward
   B) set the head reference to `null`
   C) set a reference *r* to the head of the list, and then set *r* to `null`
   D) throw an exception, because no list has a node with index 0

   Answer: A

20) A doubly linked list makes it easy to
   A) move from any node to its successor, and from any node to its predecessor
   B) skip two nodes at a time when moving forward through the list
   C) skip two nodes at a time when moving backward through the list
   D) to create a second copy of the linked list

   Answer: A

21) A circularly linked list makes it easy to
   A) move from any node to its predecessor
   B) move from any node to its successor
   C) jump from the last node to the first
   D) jump from node to node

   Answer: C

22) A doubly circularly linked list makes it easy to

A) move forward through a list, and then quickly jump to the beginning of the list when you get to the end
B) to jump from the last node to the first, from the first to the last, and to move forward and backward through the list
C) to skip two nodes at a time in either direction, and wrap around the list when you come to the end
D) do nothing: there is no such as a doubly circularly linked list

Answer: B

23) In a typical doubly linked list, a node has
A) a field to store the element, and two references to keep track of two successor nodes
B) a field to store the element, and two references to keep track of successor and predecessor nodes
C) a field to store the element, and two references to keep track of two predecessor nodes
D) either one of A or C

Answer: B

24) In a typical circular doubly linked list, a node has
A) a field to store the element, and two references to keep track of two successor nodes, and a reference to keep track of the start of the list
B) a field to store the element, and two references to keep track of successor and predecessor nodes
C) a field to store the element, and two references to keep track of two predecessor nodes, and a reference to keep track of the end of the list
D) either one of A or C

Answer: B

25) A list can be considered a recursive data structure because
A) list classes implement list interfaces
B) list objects are instances of list classes
C) if you remove the head of the list, what remains is also a list
D) None of the above: only methods can be considered recursive.

Answer: C

26) The tail of a list
A) is an instance of a class that implements the `ListTail` interface
B) is what you get when take a list and remove its head
C) is a synchronized subclass of the `LinkedList` class
D) None of the above

Answer: B

27) In many recursive operations on lists,
A) the base case of the recursion corresponds to the empty list
B) some operation that does not require recursion is performed on the head element
C) a recursive call is made on the tail of the list
D) All of the above

Answer: D

28) In order to use recursion on linked lists
A) it is necessary to modify the class that represents nodes, by adding a reference needed to support recursion
B) the class that represents nodes must implement the `Repeatable` interface
C) no changes to the node class are necessary
D) A and B are both true

Answer: C

29) When using recursion on linked lists
A) the linked list class is subclassed, and then the recursive method overrides a method of the same name in the list class
B) the recursive method should be one of the methods specified in the `List` interface
C) the recursive method should be made private, and should be called by a public non-recursive method
D) the recursive method should not call itself outside of its base case

Answer: C

30) A recursive computation of the size of a list can work as follows:

A) set a local counter to zero; loop through the list, incrementing the counter by one at each element of the list; return the counter
B) recursively compute the size of the tail, add one, and return the result
C) if the list is not empty, recursively compute the size of its tail, add one, and return the result
D) if the list is empty, return zero; otherwise, recursively compute the size of the tail, add one, and return the result

Answer: D

31) Scientists in a certain laboratory are working with a linked list class that uses recursion to compute its size. The scientists know that an empty list has size 0, so they never ask a linked list to compute its size when the list is empty. Under these circumstances,
A) the recursive method can become more efficient by eliminating the test for a base case
B) the recursive method should still handle the base case of an empty list, because it will still occur even though the scientists never ask for the size of an empty list
C) the recursive method can be modified to use a list of size 1 as the base case
D) B and C are both correct

Answer: D

32) A linked list class uses a `Node` class to represent nodes. A private recursive method

```
Node add(int index, E element, Node list)
```

takes a reference `list` (referring to the first in a chain of `Node` objects), adds a node containing the given element at the given index, and returns a reference to the first node of the resulting chain. Assume that `index` is nonnegative and is less than or equal to the size of `list`. Under these circumstances, the `add` method should handle its base case (`index` is 0) by
A) returning `list`
B) adding a node containing `element` to the front of `list` and returning a reference to the newly created node
C) adding a node containing `element` to the front of `list` and returning `list`
D) adding a node containing `element` to the end of list

Answer: B

33) A linked list class uses a `Node` class with a successor reference `next` to represent nodes. A private recursive method

```
Node add(int index, E element, Node list)
```

takes a reference `list` (referring to the first in a chain of `Node` objects), adds a node containing the given element at the given index, and returns a reference to the first node of the resulting chain. Assume that `index` is nonnegative and is less or equal to the size of `list`. Under these circumstances, the `add` method should handle its non-base case (`index` is not 0) by
A) recursively returning the value `add(index, element, list.next)`
B) setting `list.next` to `add(index-1, element, list.next)` and returning `list`
C) setting `list.next` to `add(index, element, list.next)` and returning `list`
D) recursively returning the value `add(index-1, element, list)`

Answer: B

34) A linked list class uses a `Node` class with successor reference `next`, and uses a reference `first` to point to the first node of the list. A positive index *k* is given, and we want to set a reference `pred` to point to the node with index *k*-1. The right code to use is

A)
```
pred = first;
for (int i = 0; i < k; i++)
   pred = pred.next;
```

B)
```
pred = first;
for (int i = 0; i <= k; i++)
   pred ++;
```

C)
```
pred = first;
for (int i = 1; i < k; i++)
   pred = pred.next;
```

D)
```
pred = head;
for (int k : list)
   k--;
```

Answer: C

35) A linked list class uses a `Node` class with successor reference `next` and field `element` to store values. It uses a reference `first` to point to the first node of the list. Code to print all elements stored in the list can be written as

A)
```
System.out.print(first);
```

B)
```
Node p = first;
while (p != null)
{
   System.out.println(p.element);
   p = p.next;
}
```

C)
```
Node p  = first;
while (p != null)
   System.out.println(p.next);
```

D)
```
Node p = first;
for (p!= null)
{
   System.out.println(p.element);
   p++;
}
```

Answer: B

36) A linked list class uses a `Node` class with successor reference `next` and field `element` to store values. A recursive method to print all list elements can be written as

A)
```
static void printList(Node list)
{
   if (list != null)
     {
        System.out.println(list.element);
        printList(list.next);
     }
}
```

B)
```
static void printList(Node list)
{
   while (list!= null)
    {
       System.out.println(list.element)
        printList(list.next);
        list = list.next;
     }
}
```

C)
```
static void printList(Node list)
{
   while (list.next != null)
    {
        printList(list.next);
        System.out.println(list.element)
        list ++;
     }
}
```

D)
```
static void printList(Node list)
{
   System.out.println(list.element):
   printList(list.next);
}
```

Answer: A