CIS113: Exam2

CH4-CH7

1) True or False

A recursive method should always have at least one base case.

2) True or False

Recursive algorithms are usually implemented with *while* loops.

3) True or False

The base case does not exist or is not reached when there is infinite recursion.

4-8 Consider the following method

```
int tq (int num)
{
  if (num == 0)
    return 0;
  else
  if (num > 100)
    return -1;
  else
    return num + tq ( num – 1 );
}
```

4) Is there a constraint on the value that can be passed as an argument in order for tq to pass the smaller-caller test (i.e. is there a base case)? If so, what is it?

4-8 Consider the following method

```
int tq (int num)
{
  if (num == 0)
    return 0;
  else
  if (num > 100)
    return -1;
  else
    return num + tq ( num – 1 );
}
```

5) Is tq(4) a valid call? If so, what is returned from the method?

4-8 Consider the following method

```
int tq (int num)
{
  if (num == 0)
    return 0;
  else
  if (num > 100)
    return -1;
  else
    return num + tq ( num – 1 );
}
```

6) Is tq(0) a valid call? If so, what is returned from the method?

4-8 Consider the following method

```
int tq (int num)
{
  if (num == 0)
    return 0;
  else
  if (num > 100)
    return -1;
  else
    return num + tq ( num – 1 );
}
```

7) Is tq(–5) a valid call? If so, what is returned from the method?

4-8 Consider the following method

```
int tq (int num)
{
  if (num == 0)
    return 0;
  else
  if (num > 100)
    return -1;
  else
    return num + tq ( num – 1 );
}
```

8) Is tq(250) a valid call? If so, what is returned from the method?

9-13) Match your answer

9) What is the base case?

10) What is the general case?

11) What are the constraints on the argument values?

12) An example of direct recursion?

13) An example of indirect recursion?

A. method_1 invokes method_2, which in turn invokes method_1

B. argument must not be < 0

C. argument < 0

D. method_2 invokes method_1

E. argument is equal to 0

F. argument must not be > 0

G. argument > 0

H. method_1 invokes method_1

14) Multiple Choice

Define a queue.

A. a structure in which elements are added to the top and removed from the top
B. a collection that exhibits a linear relationship among its elements
C. a structure in which elements are added to the rear and removed from the front
D. queues allow full access to elements which are not at the ends

15) True or False

A queue is a last in, first out structure.

16) True or False

If you enqueue 5 elements into an empty queue, and then perform the isEmpty operation 5 times, the queue will be empty again.

17) True or False

The enqueue operation should be classified as a "transformer".

18) True or False

In Java, an interface can extend another interface.

19-21) Match your answer

19) The *QueueUnderflowException* is potentially thrown by the following queue method(s)

20) The *QueueOverFlowException* is potentially thrown by the following queue method(s).

21) Our BoundedQueueInterface extends our

A. enqueue only
B. dequeue only
C. UnboundedQueueInterface
D. isEmpty
E. QueueInterface
F. the constructors
G. enqueue and dequeue
H. None of these

22) Show what is written by the following segment of code, given that item1, item2, and item3 are int variables, and queue is an object that fits our abstract description of a queue. Assume that you can store and retrieve variables of type int on queue. (3 points)

```
item1 = 1;
item2 = 3;
item3 = 4;
queue.enqueue(item2);
queue.enqueue(item1);
queue.enqueue(item1 + item3);
item3 = queue.dequeue();
queue.enqueue (item3*item3);
queue.enqueue(item2);
queue.enqueue(7);
item2 = queue.dequeue();
System.out.println(item1 + " " + item2 + " " + item3);
while (!queue.isEmpty())
{
  item1 = queue.dequeue();
  System.out.println(item1);
}
```

23-25) uses the following class Multiple Choice

```
package ch05.queues;

public class ArrayBndQueue<T> implements BoundedQueueInterface<T>
{
  protected final int DEFCAP = 100; // default capacity
  protected T[] queue;              // array holds queue elements
  protected int numElements = 0;    // #of elements in queue
  protected int front = 0;          // index of front of queue
  protected int rear;               // index of rear of queue

  public ArrayBndQueue()
  {
    queue = (T[]) new Object[DEFCAP];
    rear =  defCap - 1;
  }

  public ArrayBndQueue(int maxSize)
  {
    queue = (T[]) new Object[maxSize];
    rear =  maxSize - 1;
  }
```

23) Complete the implementation of the *isEmpty* method:

```
// Returns true if this queue is empty
//              otherwise returns false.
public boolean isEmpty()
{
  // complete the method body
}
```

A. return numElements;

B. return (numElements = 0);

C. return (numElements == 0);

D. return (numElements == queue.length);

23-25) uses the following class Multiple Choice

```
package ch05.queues;

public class ArrayBndQueue<T> implements BoundedQueueInterface<T>
{
  protected final int DEFCAP = 100; // default capacity
  protected T[] queue;              // array holds queue elements
  protected int numElements = 0;    // #of elements in queue
  protected int front = 0;          // index of front of queue
  protected int rear;               // index of rear of queue

  public ArrayBndQueue()
  {
    queue = (T[]) new Object[DEFCAP];
    rear =  defCap - 1;
  }

  public ArrayBndQueue(int maxSize)
  {
    queue = (T[]) new Object[maxSize];
    rear =  maxSize - 1;
  }
```

24) Find the error in the *enqueue* method

```
// Throws QueueOverflowException if this queue
//  is full, otherwise adds element to the rear of
//  this queue.
public int enqueue(T element)
{
    if (isFull())
     throw new QueueOverflowException("Enquee" +
                                    "attempted on a full queue.");
    else
    {
         rear = (rear + 1) % queue.length;
         queue[rear] = element;
          numElements = numElements + 1;
     }
     return numElements;
}
```

A. return numElements + 1;

B. return numElements – 1;

C. return (numElements == queue.length);

D. This method should have been declared void and have no return statement

E. The method is correct

23-25) uses the following class Multiple Choice

```
package ch05.queues;

public class ArrayBndQueue<T> implements BoundedQueueInterface<T>
{
  protected final int DEFCAP = 100; // default capacity
  protected T[] queue;              // array holds queue elements
  protected int numElements = 0;    // #of elements in queue
  protected int front = 0;          // index of front of queue
  protected int rear;               // index of rear of queue

  public ArrayBndQueue()
  {
    queue = (T[]) new Object[DEFCAP];
    rear =  defCap - 1;
  }

  public ArrayBndQueue(int maxSize)
  {
    queue = (T[]) new Object[maxSize];
    rear =  maxSize - 1;
  }
```

25) Find the error in the *dequeue* method

```
// Throws QueueUnderflowException if this queue is
// empty, otherwise removes front element from this
// queue and returns it.
public T dequeue()
{
   if (isEmpty())
      throw new QueueUnderflowException("Dequeue " +
                 "attempted on empty queue.");
else
   {
      T toReturn = queue[front];
      queue[front] = null;
      front = (front + 1) / queue.length;
      numElements = numElements - 1;
      return toReturn;
   }
}
```

A. In the else block queue[front] should be assigned 1 instead of null
B. front is miscalculated in the else block, should use the modulus operator
C. numElements is miscalculated in the else block, should have incremented numElements
D. The method should be void and the return statement removed.
E. if(!isEmpty()) should replace if(isEmpty())

26) True or False

When comparing two objects using the == operator what is actually compared is the contents of the objects.

27) True or False

- A list is a first in, first out structure.

28) True or False

We require our sorted lists to only contain objects that are *Comparable*. Also, we allow duplicate elements on our lists.

29) Multiple Choice

We say that a list is a collection that exhibits a linear relationship among its elements. Define linear relationship.

A. Each element has a unique predecessor.

B. Each element has a unique successor.

C. The first element does not have a predecessor.

D. The last element does not have a successor

E. Both A & B

F. Each element except the first has a unique predecessor, and each element except the last has a unique successor.

30) True or False

Our ListInterface defines the common responsibilities of all three of our list varieties: sorted, unsorted and indexed.

31-33) Classify the List Methods

31) Which of the following are iterators?
32) Which of the following are observers?
33) Which of the following are transformers?

size
contains
remove
get
toString
reset
getNext
add

34) True or False

The binary search algorithm is an efficient way for us to search our sorted lists.

35) Multiple Choice

The binary search algorithm is O(???), where N is the size of the list. Where ??? is

A. 1
B. $\log_2 N$
C. $\log_{10} N$
D. N^2
E. N^3

36-38) How many iterations?

For each of the following list sizes, show the maximum number of iterations through the binary search algorithm that would be required to search a sorted list. For extra credit explain why.

	Length of List	Number of Iterations
36)	20	_______________
37)	30	_______________
38)	1000	_______________

39) True or False

Our linked implementation of lists implement bounded lists.

40) True or False

In a circular linked list every node has a successor.

41) Multiple Choice

In our circular linked list the list element itself references

A. the first node

B. the last node

C. both the first and the last nodes

D. a list descriptor

E. none of these

42) Multiple Choice

The code for the ________________________ method is unchanged when moving from our linear list implementation to our circular list implementation.

A. find

B. reset

C. add

D. remove

E. none of these

43) Multiple Choice

The difference between a linked list and a circular linked list is:

In a circular linked list the last node is linked to the first node, whereas in a linked list the last node has a null link.

44) True or False

In our doubly linked list the nodes are linked in two directions, front and back.

45) True or False

The purpose of header and trailer nodes in a list is:

They can simplify processing by removing the need to code for special cases when dealing with the first or last nodes on a list.

46) True or False

It is possible to use an array to implement a linked list.

47) Multiple Choice

In our array based linked list implementation we use the value __________ to indicate an empty link.

A. -1

B. 0

C. maxElements

D. null

E. none of these

48) True or False

- The efficiency of the Selection, Insertion, and Bubble sort algorithms is $O(N^2)$ where N is the size of the list being sorted.

49) Short answer

Given that the order of the following list after two iterations of the “inner” part of the Selection Sort algorithm?

13, 4, 16, 19, 2, 15, 12, 3, 23, 20

is

2, 3, 16, 19, 13, 15, 12, 4, 23, 20

What would be the order of the following list after seven iterations of the “inner” part of the Selection Sort algorithm?

50) Short answer

Given that the order of the following list after two iterations of the “inner” part of the Bubble Sort algorithm?

13, 4, 16, 19, 2, 15, 12, 3, 23, 20

is

2, 3, 13, 4, 16, 19, 12, 15, 20, 23

What would be the order of the following list after seven iterations of the “inner” part of the Selection Sort algorithm?

10, 9, 8, 7, 6, 5, 4, 3, 2, 1