Question 1. [10 MARKS]

No sense at all: give 0 or 1 depending on amount and odor of crap.

A)

-2 no call to grfn()
-3 if grfn logic done in hasNext()
-1 grfn before storing current result
-1 doesn't check whether hasAnotherSentence
-2 doesn't throw exception
-0.5 if it doesn't append '.'
-1 no return
-1 not preserving next sentence in a variable.
-1 treating a primitive like a class
-2 making up a method to do whatever they can't figure out
-1 private on local vars
-0.5 problem with scope of local variable

B)

-1 no setup for hasNext
-1 private on local vars
-2 doesn't throw exception
-1 treating a primitive like a class
-2 making up a method to do whatever they can't figure out
-1 if set condition to -1 to terminate but check for -1 for EOF
-1 extraneous while loop
-1 invalid value at loop entry
-1 return String from grfn (it's void)
-0.5 uses '.' instead of SENTENCE_TERMINATOR
-0.5 uses || instead of &&
-1 doesn't use Reader.read
-1 terminate with only one condition
-1 not setting hasAnotherSentence when return value on read is -1
-3 no while loop
-0.5 problem with scope of local variable

```
import java.io.*;
import java.util.*;

/**
 * An iterator for English sentences. A sentence is a sequence of characters
 * ending with a period.
 */
public class SentenceIterator implements Iterator {

  /** Where the sentences come from. */
  private Reader reader;

  /** The next sentence to be returned. */
  private String nextSentence;

  /** The character marking the end of a sentence. */
  public static final char SENTENCE_TERMINATOR = '.';

  /** True if there is another sentence; false otherwise. */
  public boolean hasAnotherSentence = true;

  /**
   * An iterator for the sentences in r.
   * Requires: r != null;
   */
  public SentenceIterator(Reader r) {
    reader = r;
    getReadyForNext();
  }

  /**
   * Return whether there is another sequence of characters ending with a period.
   */
  public boolean hasNext() {
    return hasAnotherSentence;
  }

  /**
   * Return the next sentence: a sequence of characters ending with a period.
   * @throws NoSuchElementException if there are no more sentences.
   */
  public Object next() {
    if (!hasNext()) {
      throw new NoSuchElementException();
    }

    String result = nextSentence;
    getReadyForNext();
    return result;
  }

  /**
   * Read the next sequence of characters into nextSentence. Set
   * hasAnotherSentence to false if either the end of the input
```

```
   * stream is reached (that is, if read() returns -1), or if an
   * IOException is thrown by read().
   */
  private void getReadyForNext() {
    nextSentence = "";
    try {
      int i = reader.read();
      while (i != -1 && (char) i != SENTENCE_TERMINATOR) {
        nextSentence += (char) i;
        i = reader.read();
      }

      if ((char) i != SENTENCE_TERMINATOR) {
        hasAnotherSentence = false;
      } else {
        nextSentence += SENTENCE_TERMINATOR;
      }
    } catch (IOException e) {
      hasAnotherSentence = false;
    }
  }

  /**
   * Throw an UnsupportedOperationException.
   */
  public void remove() {
    throw new UnsupportedOperationException();
  }
}
```

Question 2. [10 MARKS]

Part (a) [5 MARKS]

```
public static Node mergesort(Node head) {

  if (head == null || head.next == null) {
    return head;
  }

  Node temp = head.next;
  Node middle = head;
  while (temp != null) {

  // FILL IN MISSING LINES HERE


    temp = temp.next;
    if (temp != null) {
       temp = temp.next;
       middle = middle.next;
    }

  }

  temp = middle;
  middle = temp.next;
  temp.next = null;
  return merge(mergesort(head), mergesort(middle));
}
```

Question 3. [10 MARKS]

```
public class JumpList {

    /** A node in the JumpList. */
    private static class Node {

        /** The value in the node. */
        public int data;

        /** The next and previous pointers. */
        public Node next, prev;

        /** Pointers to nodes several steps to the right and left. */
        public Node jump, backjump;
    }

    /** The first and last nodes in the list, or null if there are none. */
    private Node head, tail;

    YOUR CODE WILL GO HERE
}
```

Part (a) [5 MARKS]

MARKING

```
-2.5 Just using next and not jump
-1 q is not used or used improperly.
-2 boolean variable declared but used incorrectly.
-1/2 solution correct but misses requirement (for each requirement)
-0.5 Second loop test: missing either q.data < k or q != null
-0.5 missing p!= null && p.data == k in return
-0.5 missing q != null check in return
```

```
/**
 * Return true if k is an element in this list, and false otherwise.
 * @param k The value sought.
 */
 public boolean contains(int k) {

   Node p = head; // the current node being examined
   Node q = null; // the node last examined, or null if p == head.

   while (p != null && p.data < k) {
     q = p;
     p = p.jump;
   }

   while (q != null && q.data < k) {
     q = q.next;
   }

   return (q != null && q.data == k)
     || (q == null && p != null && p.data == k);
   }
 }
```

Part (b) [5 MARKS]

```
/**
 * Return true if and only if the jump list referred to by front satisfies
 * the two jump list properties, and if the prev pointers are set correctly.
 *
 * Requires: front != null and front.prev == null.
 */
 public static boolean isValidJumpList(Node front) {
```

Question 4. [10 MARKS]

Part (a) [3 MARKS] MARKING

```
-2 for changing r.left != null case.
-2 for changing result += r.key line.
-2 for not changing r.right != null case.
Special case: -1 degenerate case, e.g.
                 result += r.key + " ";
                 result += toString(right);
```

Change the last line to:

```
      if (r.right != null) {
        result += " " + toString(r.right);
      }
```

Part (b) [5 MARKS]

MARKING

```
-1 each kind of syntactic error
-4 no recursion
-2 Each flaw with setup.
-2 Right setup but problem with recursion.
   Some people weren't returning the recursive call, etc.

-1 Not handling the 'c is in the tree' case.
```

```
  /** Return a pointer to the parent of the node that will contain c
   * in the subtree rooted at r, or null if c is already in the tree.
   * Requires: r != null.
   * @param c the key to find.
   * @return the parent of the node that will be the parent of c,
   * or null if c is in this tree.
   */
  public static BSTNode findParent(BSTNode r, Comparable c) {

    BSTNode t = r; // assume r is the parent.
    if (c.compareTo(r.key) < 0) {
      if (r.left != null) {
        t = findParent(r.left, c);
      }
    } else if (c.compareTo(r.key) > 0) {
      if (r.right != null) {
        t = findParent(r.right, c);
      }
    } else {
```

```
      // r contains c.
      t = null;
    }

    return t;
  }
```

Part (c) [2 MARKS]

MARKING

```
-2 incorrect
-1 poorly worded or unclear
```

Correct answers:

```
    - it reveals the implementation.
    - TreeNode is private so users of the class will be confused.
```

Question 5. [10 MARKS]

Part (a) [4 MARKS]

Complete the following **recursive** method.

```
/** Calculate the number of coins in a coin pyramid with numLevels levels. */
public static int numCoins(int numLevels) {


  if (level == 0) {
    return 0;
  } else {
    return level * level + numCoins(level - 1);
  }
}
```

MARKING:

```
-2 no recursion
-1 no n==0 base case
-1 doesn't add correct value to total
-1 doesn't return anything
-1 recursive call no stored
-2 no base case/recursive step
-1 inexplicable behaviour
```

Part (b) [3 MARKS]

Complete the following **iterative** method to calculate the number of coins in the pyramid:

```
/** Calculate the number of coins in a coin pyramid with numLevels levels. */
public static int numCoins(int numLevels) {

  int sum = 0;
  for (int i = 1; i <= numLevels; i++) {
    sum += i * i;
  }

  return sum;
}
```

MARKING:

```
-1 using wrong bounds for loop
-1 omitting zero case
-2 using recursion
-1 using iterators
-1 ignoring numLevels
-1 inexplicable behaviour
```

Part (c) [3 MARKS]

Complete the following method. You may make use of method `numCoins` from the previous parts even if you did not complete it.

```
/**
 * Print on separate lines:
 * - the height of the largest pyramid that could be built with n coins
 * - the total value of that pyramid assuming each coin is worth coinValue cents
 * - the change left over.
 */
public static void printCoinInfo(int n, int coinValue) {
  int height = 0;
  while (numCoins(height) < n) {
    height++;
  }

  System.out.println(height);
  System.out.println(numCoins(height) * coinValue);
  System.out.println(n - numCoins(height) * coinValue);


}
```

MARKING:

```
-1 making height one level too high
-1 not printing out/calculating a value correctly
-1 not finding height accurately
-1 omitting zero case
-2 using sqrt or log to find height
-1 inexplicable behaviour
```

Question 6. [10 MARKS]

MARKING

```
a: n

b: log_26 n (-1, +1 are okay, ceiling okay); no base: 0.5; base 2: 0;
   log n / log 26 okay

c: k^n; if 26^n: 0.5

d: O(m * n); if no O() notation then 0.5

e: x^2; if O(x^2) 0.5

f: 2xy; if O(2xy) 0.5
```

Part (a) [1 MARK]

Answer: n

Part (b) [1 MARK]

Answer: $log_{26}n$

Part (c) [1 MARK]

Answer: k^n

Part (d) [1 MARK]

Answer: $\mathrm{O}(m * n)$

Part (e) [1 MARK]

Answer: x^2

Part (f) [1 MARK]

Answer: $2 * x * y$

Part (g) [4 MARKS]

MARKING

```
Okay to skip O(1) part.
-1 for each missing part.
-2 not using right analysis style.
-1 for n(n2+n)/2
```

```
    for (int i = 1; i <= n; i++) {
        int[] array = new int[i];
        for (int col = 0; col < i; col++) {
            array[col] = doExpensiveThing();
        }
    }
```

Total Marks = 60