1. (10 points) Below is a recursive drawing. The figure on the left shows several levels of recursion, while the figure on the right shows a drawing made with just a single recursive call.

This drawing is made by first drawing an H shape and then drawing 4 smaller H’s attached to the bigger H. Each of the smaller H’s is half the size of its predecessor. Here is a method to draw an individual H shape:

```java
/** Draw an H shape
   * @param x the center x value
   * @param y the center y value
   * @param size the length of one line in the H
   * @param g the graphics object to draw on
   */
private void drawHShape(int x, int y, int size, Graphics g) {
    int left = x - size/2;
    int right = x + size/2;
    int top = y - size/2;
    int bottom = y + size/2;

    // Draw the horizontal line
    g.drawLine (left, y, right, y);

    // Draw the left vertical line
    g.drawLine (left, top, left, bottom);

    // Draw the right vertical line
    g.drawLine (right, top, right, bottom);
}
```

Complete the method on the following page to draw the recursive shape, using the drawHShape method shown above. Stop drawing when the size is less than 20.
public void drawHTree (int centerX, int centerY,
    int lineLength, Graphics g) {

}
2. Suppose you are using a linked list to maintain elements in sorted order. Whenever you add an element to the list, you will walk the list looking for the element with the next higher value and insert the new element there. Below is some code intended to do that. You can assume that the code compiles with no errors.

```java
public void insertInOrder(Integer item) {
    Node<Integer> last = null;
    Node<Integer> current = head;
    while (current != null && current.getValue() < item) {
        last = current;
        current = current.getNext();
    }
    Node<Integer> newNode = new Node<Integer> (item);
    last.setNext(newNode);
    newNode.setPrev(last);
    newNode.setNext(current);
    current.setPrev(newNode);
    if (current == null) {
        tail = newNode;
    }
    if (last == null) {
        head = newNode;
    }
}
```

For each part of this question, assume that the list has the following state when insertInOrder is called. That is, parts b & c start with the same original list, not the list that results from the calls performed in part a or b.

![Diagram of a linked list with nodes 1, 3, and 5](image)

a. (5 points) What happens when calling insertInOrder with 2 as the value for the parameter?

b. (5 points) What happens when calling insertInOrder with 7 as the value for the parameter?

c. (5 points) What happens when calling insertInOrder with 0 as the value for the parameter?
3. (5 points) What output does the following program produce? You can assume that the program compiles and executes without producing any errors.

```java
public class Mystery {

    private void mysteryMethod(ArrayList<String> s) {
        Stack<String> words = new Stack<String>();
        Iterator<String> iter = s.iterator();

        while (iter.hasNext()) {
            words.push(iter.next());
        }

        String result = "";
        while (!words.isEmpty()) {
            result = result + " " + words.pop();
        }

        System.out.println(result);
    }

    public static void main (String[] args) {
        ArrayList<String> s = new ArrayList<String>();
        s.add("here");
        s.add("comes");
        s.add("sumemr");

        Mystery mystery = new Mystery();

        mystery.mysteryMethod(s);
    }
}
```
4. Each of the following methods creates a string consisting of n x’s. Recall that String concatenation (the + operator) copies both of the String arguments when creating the new String. In contrast, StringBuilder’s append method only copies the parameter passed to append. The toString method for StringBuilder copies the entire String. What is the O() time of each of the following methods. Briefly explain your answers.

a. (5 points)
   public String m1 (int n) {
      String s = "";
      for (int i = 0; i < n; i++) {
         s = s + "x"
      }
      return s;
   }

b. (5 points)
   public String m2 (int n) {
      // The constructor call is O(n)
      StringBuilder sb = new StringBuilder(n);

      for (int i = 0; i < n; i++) {
         sb.append("x");
      }
      return sb.toString();
   }
5. (5 points) The following program is incorrect. When the user runs the program, the following appears in the Console:

0
Exception in thread "main" java.lang.NullPointerException
    at Bug.<init>(Bug.java:15)
    at Bug.main(Bug.java:27)

Correct the code shown below so that it stores the even numbers from 0 to 18 in the evenNumbers array and does not show a stack trace when executed. I have added line numbers to help you; they are not part of the Java.

1. public class Bug {
2.     private int[] evenNumbers;
3. 
4. 
5. 
6.     public Bug () {
7. 
8.     
9.         for (int i = 0; i < 10; i++) {
10.            int nextEvenNumber = i * 2;
11.            evenNumbers[i] = nextEvenNumber;
12.     }
13. 
14. 
15.     }
16. 
17. 
18. }
19. 
20. 
21. }
22. 
23. 
24. public static void main (String[] args) {
25. 
26.     new Bug();
27. 
28. 
29. }
30. }
6. Please answer the questions below about this program. You can assume that it compiles and executes without any errors.

```java
public class Mystery {
    private Queue<Integer> q1;
    private Queue<Integer> q2;

    public Mystery(Queue<Integer> q1, Queue<Integer> q2) {
        this.q1 = q1;
        this.q2 = q2;
    }

    private Queue<Integer> mysteryMethod() {
        Queue<Integer> resultQueue = new LinkedList<Integer>();
        while (!q1.isEmpty() && !q2.isEmpty()) {
            if (q1.peek() <= q2.peek()) {
                resultQueue.enqueue(q1.dequeue());
            } else {
                resultQueue.enqueue(q2.dequeue());
            }
        }

        while (!q1.isEmpty()) {
            resultQueue.enqueue(q1.dequeue());
        }

        while (!q2.isEmpty()) {
            resultQueue.enqueue(q2.dequeue());
        }

        return resultQueue;
    }

    public static void main(String[] args) {
        Queue<Integer> q1 = new LinkedList<Integer>();
        q1.enqueue(1);
        q1.enqueue(2);
        q1.enqueue(4);

        Queue<Integer> q2 = new LinkedList<Integer>();
        q2.enqueue(3);
        q2.enqueue(5);

        Mystery m = new Mystery(q1, q2);
        Queue<Integer> q3 = m.mysteryMethod();
        Iterator<Integer> iter = q3.iterator();
        while (iter.hasNext()) {
            System.out.print(iter.next() + " ");
        }
        System.out.println();
    }
}
```
a. (5 points) What value does q1 have before the mystery method is called? (Show the contents of the linked list like we have done in class, drawing the nodes linked together.)

b. (5 points) What value does q2 have before the mystery method is called?

c. (5 points) What output does the main method print?
7. (7 points) Here is a class that compiles and executes. However, some of its instance variables should be local to methods instead. Modify the code below to show the local variable declarations where appropriate.

```java
public class WebBrowser extends JPanel implements ActionListener {

private JPanel urlPanel;
private JTextField urlField = new JTextField(50);
private JTextArea pageArea = new JTextArea(30, 50);
private JScrollPane pageScroller;
private String url;
private Scanner pageIn;
private String curLine;

/** Creates the user interface for this simple Web browser */
public WebBrowser() {
    setLayout(new BorderLayout());

    urlPanel = new JPanel();
    urlPanel.add(new JLabel("URL:"));
    urlPanel.add(urlField);
    urlField.addActionListener(this);
    add(urlPanel, BorderLayout.NORTH);

    pageScroller = new JScrollPane(pageArea);
    add(pageScroller, BorderLayout.CENTER);
}

/** Loads the Web page when the user hits in the enter key in the URL field. */
public void actionPerformed(ActionEvent e) {
    url = urlField.getText();
    try {
        // Open the URL for reading.
        pageIn = new Scanner(new URL(url).openStream());

        // Read each line and add it to the display
        while (pageIn.hasNextLine()) {
            curLine = pageIn.nextLine() + "\n";
            pageArea.append(curLine);
        }

        pageIn.close();
    } catch (Exception exc) {
        JOptionPane.showMessageDialog(this, "Could not load Web page at " + url);
    }
}

```
8. (10 points) Complete the `convertToString` method below, writing it recursively. Recall that you can use the `%` (mod) operator to get a remainder and `/` (integer division) to get division that ignores the remainder. For example, 123 / 10 = 12 and 123 % 10 = 3. Be sure to handle the case of negative numbers as well.

```java
public class RecursiveIntToString {
    char[] chars = new char[10];

    public RecursiveIntToString() {
        chars[0] = '0';
        chars[1] = '1';
        chars[2] = '2';
        chars[3] = '3';
        chars[4] = '4';
        chars[5] = '5';
        chars[6] = '6';
        chars[7] = '7';
        chars[8] = '8';
        chars[9] = '9';
    }

    public String convertToString(int num) {
        // Your recursive code here
    }
}
```
9. (2 points each) Another way to represent a long list is to break it into a collection of chunks. Each chunk can hold a limited number of values, but there can be an unlimited number of chunks. For example, if a chunk could hold 1000 values, and you wanted to store 10,000 values, you would need to have 10 chunks. Below is a partial implementation of such a data structure, including only add operations. Please answer the questions that follow about the performance of the implementation. If you use n in the O() function (like O(n), for example), be sure to indicate what n refers to (the size of the list, the size of the chunk, or the number of chunks in the list).

```java
public class LargeList<T> {
    public static final int CHUNK_SIZE = 1000;
    private ArrayList<Chunk<T>> chunks = new ArrayList<Chunk<T>>();

    public void add(T newValue) {
        Chunk<T> lastChunk;

        if (chunks.size() == 0) {
            lastChunk = addChunk();
        } else {
            lastChunk = chunks.get(chunks.size() - 1);
            if (lastChunk.size() == CHUNK_SIZE) {
                lastChunk = addChunk();
            }
        }
        lastChunk.add(newValue);
    }

    private Chunk<T> addChunk() {
        Chunk<T> newChunk = new Chunk<T>();
        chunks.add(newChunk);
        return newChunk;
    }
}
```
public void add (int index, T newValue) {
    int whichChunk = index / CHUNK_SIZE;
    int chunkIndex = index % CHUNK_SIZE;

    if (index < 0) {
        throw new IndexOutOfBoundsException();
    }

    Chunk<T> chunkToChange;
    if (whichChunk == chunks.size()) {
        if (chunkIndex != 0) {
            throw new IndexOutOfBoundsException();
        }
        else {
            chunkToChange = addChunk();
        }
    }
    else {
        chunkToChange = chunks.get(whichChunk);
    }

    T extraValue = chunkToChange.add(chunkIndex, newValue);
    for (int chunkNum = whichChunk+1; chunkNum < chunks.size(); chunkNum++) {
        extraValue = chunks.get(chunkNum).add(extraValue);
    }
    if (extraValue != null) {
        Chunk<T> newChunk = addChunk();
        newChunk.add(extraValue);
    }
}

private static class Chunk<T> {
    private LinkedList<T> values = new LinkedList<T>();

    public int size() {
        return values.size();
    }

    public T add (T value) {
        values.add(value);
        if (values.size() > CHUNK_SIZE) {
            return values.remove(values.size()-1);
        }
        return null;
    }
}
public T add (int index, T value) {
    values.add(index, value);
    if (values.size() > CHUNK_SIZE) {
        return values.remove(values.size()-1);
    }
    return null;
}
}
}

a. What is the O() of the add(T value) method in the Chunk class?

b. What is the O() of the add(int index, T value) method in the Chunk class?

c. What is the O() of the addChunk method in the LargeList class?

d. What is the O() of the add(T value) method in the LargeList class?

e. What is the O() of the add(int index, T value) method in the LargeList class?
10. (3 points each) Answer the questions below about the following classes and interfaces.

```java
interface Sport {
    public boolean usesABall();
    public boolean isPlayedIndoors();
}

abstract class SnowSport implements Sport {
    public boolean isPlayedIndoors() {
        return false;
    }
}

class Skiing extends SnowSport {
    public boolean usesABall() {
        return false;
    }

    public boolean usesSkis() {
        return true;
    }
}

class SnowBallFight extends SnowSport {
    public boolean usesABall() {
        return true;
    }

    public boolean usesSkis() {
        return false;
    }
}
```

For each of these questions if the statement would compile indicate what value it would output. If it would not compile, briefly explain why it would not compile. (You can assume that the only possible compilation errors are related to polymorphism, not things like missing semicolons, wrong numbers of parameters, etc.).
a. Sport s = new Sport();
   System.out.println(s.usesABall());

b. SnowBallFight f = new SnowBallFight();
   System.out.println(f.usesABall());

c. SnowSport s = new Skiing();
   System.out.println(s.usesABall());

d. Sport s = new Skiing();
   System.out.println(s.usesSkis());

e. Sport s = new Skiing();
   SnowBallFight f = (SnowBallFight) s;
   System.out.println(f.usesSkis());
11. (2 points each) Please answer the following true/false questions. If your answer is false, give a brief explanation of why it is false.

a. A queue is a Last-In-First-Out data structure.

b. A stack is a Last-In-First-Out data structure.

c. LinkedList is a subclass of ArrayList.

d. An algorithm whose performance is $O(n)$ is less efficient than an algorithm whose performance is $O(\log n)$. (Assume the two algorithms have the same functionality.)

e. A class can implement more than one interface.

f. A class can extend more than one class.

g. A breakpoint is the point in your program where there is a bug.

h. JUnit tests compare expected values with computed values to determine if a method computes the correct answer.

i. An exception can be thrown in one method and handled in the method that called it.

j. Calling nextInt() on a Scanner object two times in a row will always return the same value.
12. (10 points) Below is the implementation for an Iterator for an ArrayList. Please complete the reverseIterator method shown below. This iterator should return the values in reverse order (that is, from the highest index to the lowest).

```java
public class ArrayList<T> {
    // Note: Java does not allow an array to have a generic type, like T[]
    private Object[] values;
    private int numValues;

    // Implementations of ArrayList methods omitted.

    public Iterator<T> iterator() {
        return new ArrayListIterator<T>();
    }

    private class ArrayListIterator<T> implements Iterator<T> {
        private int next = 0;

        public boolean hasNext() {
            return next < numValues;
        }

        public T next() {
            T returnValue = (T)values[next];
            next++;
            return returnValue;
        }
    }

    public Iterator<T> reverseIterator() {
    }
}
```