Question 1. [14 marks]

For each of the following operations briefly describe the most efficient (in the big-O sense) algorithm.
Give the worst-case runtime efficiency in big-O (using the smallest, simplest expression).

Part (a) [2 marks] Given a balanced binary search tree of $n$ nodes containing integers, return true if it contains the number 32.
Algorithm: Binary search
Runtime efficiency: $O(\log(n))$

Part (b) [2 marks] Given a linked list of $n$ nodes containing integers, move the node with the largest value to the end of the list.
Algorithm: Loop through list to find previous node to largest item. Move largest item.
Runtime efficiency: $O(n)$

Part (c) [2 marks] Given two arrays of $n$ integers, check if every integer in the first array also appears in the second array.
Algorithm: Sort both arrays, then use an algorithm similar to merge.
Runtime efficiency: $O(n \log(n))$

Part (d) [2 marks] Given a balanced binary search tree of $n$ nodes containing integers, return the biggest difference between any node and its successor.
Algorithm: Do an in-order traversal, keeping track of the biggest gap so far.
Runtime efficiency: $O(n)$

Part (e) [2 marks] Given a sorted array of $n$ integers, return the smallest integer in the array which is larger than 42 (if any).
Algorithm: Do a binary search for 42, remember the last index checked. The value to be returned is either at that index or at the next index (if any).
Runtime efficiency: $O(\log(n))$

Part (f) [2 marks] Given an array of $n$ integers, determine if there are two elements in this array which sum to 100.
Algorithm: Sort the array $O(n \log(n))$. For each item, use binary search to search for $100 - \text{item}$.
Runtime efficiency: $O(n \log(n))$

Part (g) [2 marks] Given an array of $n$ integers, check if there is a subset of this array whose elements sum to 0.
Algorithm: Recursively generate the list of sums of all subsets of the first $k$ items. This list has length $\leq 2^k$. To process the $(k + 1)^{st}$ item, append the sum of the $(k + 1)^{st}$ item with each of the previous sums to the list (runtime $O(2^k)$). Return true if 0 ever appears as a sum.
Runtime efficiency: $O(2^n)$
Question 2.  [14 marks]

Complete the method has100Run below. The only objects you may create are instances of the class CircularQueue from the lectures. This class has the constructor CircularQueue(int capacity) and implements the following interface:

```
public interface Queue {
    void enqueue(Object o);
    Object head();
    Object dequeue();
    int size();
    int capacity();
}
```

Recall the Integer class has a method int intValue() which returns the value. Also recall the Iterator interface has the methods:

```
public interface Iterator {
    Object next();
    boolean hasNext();
}
```

/** Returns whether i has a contiguous (i.e. with no gaps) subsequence of elements * whose values add to exactly 100.  */
/* Precondition: i produces only Integer objects, with strictly positive values. */
public static boolean has100Run(Iterator i)

    int sum = 0;
    Queue q = new CircularQueue(100);

    while (i.hasNext()) {
        // Loop Invariant: sum == sum of elements in queue.
        while (i.hasNext() && sum < 100) {
            int val = (Integer) i.next(); // uses auto-unboxing
            q.enqueue(val); // uses auto-boxing
            sum += val;
        }
        while (sum > 100) {
            // q cannot be empty when sum > 0.
            int val = (Integer) q.dequeue(); // uses auto-unboxing
            sum -= val;
        }
        if (sum == 100) {
            return true;
        }
    }

    return false;
}
Question 3.  [12 marks]

Consider the following recursive method:

```java
public static int m(int a, int b, int c) {
    if (a > b) {
        return m(b, a, c);
    } else if (b > c) {
        return m(a, c, b);
    } else {
        return c;
    }
}
```

Part (a)  [4 marks]

Give a good Javadoc method comment for m.

```java
/** 
 * Returns the maximum of the three integer arguments. 
 * @param a first integer 
 * @param b second integer 
 * @param c third integer 
 * @return the maximum of the three integers. */
```

Part (b)  [2 marks]

Give an example of concrete values a, b and c for which m(a, b, c) results in as deep a recursion as possible.

Answer: (a, b, c) = (3, 2, 1)

Part (c)  [6 marks]

Show the runtime stack after calling m with your values of a, b and c, just before return c executes for the first time.
Question 4. [14 marks]

Consider the following class for nodes in a Linked List:

```java
class ListNode {
    public Comparable data;
    public ListNode link;
}
```

Complete the following method. You must not use recursion. You can use non-recursive helper methods. Do not use any other datatypes such as arrays.

Recall that the Comparable interface consists of the method int compareTo(Object o).

```java
/** Merge two linked lists that are sorted in non-decreasing order into one
 * sorted list containing all of the elements in the two given lists,
 * including any duplicate elements. Returns the head of the merged list.
 * This method reuses the nodes in the original two lists.
 * @throws java.lang.IllegalArgumentException (a RuntimeException) if
 * either argument list h1 or h2 is not sorted. */
public static ListNode merge(ListNode h1, ListNode h2) {

    ListNode prev = null;
    ListNode head = null;

    while (h1 != null && h2 != null) {
        if (h1.data.compareTo(h2.data) <= 0) {
            if (prev == null) {
                head = h1;
            } else {
                prev = h1;
                h1 = h1.link;
            }
        } else {
            prev = h2;
            h2 = h2.link;
        }
    }

    // At least one of h1 and h2 is null.
    // Continued on next page...
```
// The following does not check the remaining list is sorted.
if (h1 != null) {
    if (head == null)
        head = h1;
    else
        prev.link = h1;
}
if (h2 != null) {
    if (head == null)
        head = h2;
    else
        prev.link = h2;
}
return head;
}
Question 5. [10 marks]

Consider the following class for nodes in a Binary Search Tree:

class BSTNode {
  public Comparable key;
  public BSTNode left;
  public BSTNode right;
}

Complete the following method. Do not use recursion. You can use helper methods, but they cannot be recursive. Do not use any other datatypes such as arrays or lists.

Recall that the Comparable interface consists of the method int compareTo(Object o).

/** Return the second smallest key in the binary search tree rooted at t. 
   * Precondition: t contains at least two keys, and all keys are distinct. */
public static Comparable secondSmallest(BSTNode t) {

  BSTNode prev = null;
  BSTNode sm = t;
  // Follow left links to smallest.
  while (sm.left != null) {
    prev = sm;
    sm = sm.left;
  }
  if (sm.right != null)
    return smallest(sm.right);
  else
    return prev.key;
}

public static Comparable smallest(BSTNode s) {
  while (s.left != null) {
    s = s.left;
  }
  return s.key;
}
Question 6.  [14 marks]

Consider the following class for nodes in a Binary Search Tree:

class BSTNode {
    public Comparable key;
    public BSTNode left;
    public BSTNode right;
}

Complete the following method. You can use a helper methods. You must use recursion. Do not use any other datatypes such as arrays or lists.

Recall that the Comparable interface consists of the method int compareTo(Object o).

    /** Return true exactly when the tree rooted at t is a binary search tree.
     * Precondition: All the keys in the tree are distinct. */
    public static boolean isBST(BSTNode t) {
        if (t == null)
            return true;
        return isBSTHelper(t, null, null);
    }

    /** Recursively check that the subtree root t has a key between the min
     * (mn) and max (mx) values of its ancestors. */
    public boolean isBSTHelper(BSTNode t, Comparable mn, Comparable mx) {
        boolean check = true;
        if ((mn != null) && (t.key.compareTo(mn) < 0))
            check = false;
        if ((mx != null) && (t.key.compareTo(mx) > 0))
            check = false;
        if (check && t.left != null)
            check = isBSTHelper(t.left, mn, t.key);
        if (check && t.right != null)
            check = isBSTHelper(t.right, t.key, mx);
        return check;
    }
For scratch work.

Total Marks = 78