1. ArrayList Mystery.

a) `[1, 3, 20, 40]`
b) `[3, 20, 7, 40, 80]`
c) `[20, 1, 30, 60, 40, 80]`

2. Collections Programming.

```
public static int examMode(Map<String, Integer> m, Set<String> s) {
   Map<Integer, Integer> counts = new HashMap<Integer, Integer>();
   for(String str : m.keySet()) {
      if(s.contains(str.toLowerCase())) {
         int score = m.get(str);
         if(!counts.containsKey(score)) {
            counts.put(score, 1);
         } else {
            counts.put(score, counts.get(score) + 1);
         }
      }
   }

   int mode = 0;
   int count = -1;
   for(Integer i : counts.keySet()) {
      int current = counts.get(i);
      if(current > count || (current == count && i > mode)) {
         mode = i;
         count = counts.get(i);
      }
   }
   return mode;
}
```

3. Stack and Queue Programming.

```
public static void interleaveReverse(Stack<Integer> s) {
   if(s == null) {
      throw new IllegalArgumentException();
   }
   Queue<Integer> q = new LinkedList<Integer>();
   s2q(s, q);

   int size = q.size();
   for(int i = 0; i < size; i++) {
      int current = q.remove();
      s.push(current);
      q.add(current);
   }
   for(int i = 0; i < size; i++) {
      q.add(s.pop());
      q.add(q.remove());
   }
   q2s(q, s);
}
```

4. Linked Nodes.

a)

```
ListNode temp = p;
p = p.next;
p.next.next = temp;
temp.next = null;
```

```
b)
p.next = q.next;
q.next = p;
q = null;
```

```
c)
q.next.next.next = p.next.next;
ListNode temp = q.next;
q.next = q.next.next;
temp.next = p.next;
temp.next.next = p;
p.next = null;
p = temp;
```

```
d)
ListNode temp = p.next;
p.next = p.next.next;
p.next.next = q.next;
q.next = q.next.next;
temp.next = q;
q = temp;
p.next.next.next = null;
```

5. Recursive Tracing.

a) 0
b) 8
c) 20
d) 305
e) 704

6. Recursive Programming.

```
public String replace(String s, char target, char replacement) {
      if (s.length() == 0) {
            return "";
      } else {
            String rest = replace(s.substring(1), target, replacement);
            if (s.charAt(0) == target) {
                  return replacement + rest;
            } else {
                  return s.charAt(0) + rest;
            }
      }
}
```