Questions 13 and 14 refer to the following class and declaration.

```
public class DigitalClock
{
    //private instance variables and constructors not shown
        ...

    //Returns true if this DigitalClock is defective, false otherwise.
    public boolean isDefective()
    { /* implementation not shown */ }

    //Mutator method. Advances the DigitalClock by one minute.
    public void advanceTime()
    { /* implementation not shown */ }

    //other methods not shown ...
}
```

The declaration below occurs in a client class.

```
ArrayList<DigitalClock> clocks = new ArrayList<DigitalClock>();
```

```
ArrayList<DigitalClock> clocks = new ArrayList<DigitalClock>();
```

13. Suppose that the `ArrayList` `clocks` has been initialized and contains a nonempty list of `DigitalClock` objects. Which of the following code segments will correctly remove all defective clocks?

I
```
for (DigitalClock c : clocks)
{
    if (c.isDefective())
        c.remove();
}
```

II
```
for (int index = 0; index < clocks.size(); index++)
{
    if (clocks.get(index).isDefective())
        clocks.remove(index);
}
```

III
```
Iterator<DigitalClock> itr = clocks.iterator();
while (itr.hasNext())
{
    if (itr.next().isDefective())
        itr.remove();
}
```

```
{
    if (c.isDefective())
        c.remove();
}
```

II
```
for (int index = 0; index < clocks.size(); index++)
{
    if (clocks.get(index).isDefective())
        clocks.remove(index);
}
```

III
```
Iterator<DigitalClock> itr = clocks.iterator();
while (itr.hasNext())
{
    if (itr.next().isDefective())
        itr.remove();
}
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

14. Again, suppose that `clocks` is initialized. Assume that it contains a nonempty list of correctly functioning `DigitalClock` objects. Which of the following code segments will correctly advance the time on every digital clock?

I
```
for (DigitalClock c : clocks)
{
    c.advanceTime();
}
```

II
```
for (int index = 0; index < clocks.size(); index++)
{
    clocks.get(index).advanceTime();
}
```

III
```
Iterator<DigitalClock> itr = clocks.iterator()
while (itr.hasNext())
{
    itr.next().advanceTime();
}
```

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I, II, and III

15. A large charity organization maintains a database of its donors. For each donor, the following information is stored: name, address, phone number, amount and date of most recent contribution, and total contributed so far. Two plans for organizing and modifying the data are considered:

I A one-dimensional array of `Donor` objects maintained in alphabetical order by name.

II A hash table of `Donor` objects implemented using an array of linked lists. The hash address for any given `Donor` object will be determined by a hash method that uniformly distributes donors throughout the table.

Which of the following is *false*? (Assume the most efficient algorithms possible.)

(A) Plans I and II have roughly the same memory efficiency.

(B) Insertion of a new donor is more run-time efficient using plan II.

(C) Modifying an existing donor's record is more run-time efficient using plan II.

(D) Printing out a mailing list in alphabetical order is more run-time efficient using plan I.

(E) Printing out a list of donors in decreasing order of total amount contributed is more run-time efficient using plan I.

16. Consider a class that has this private instance variable:

```
private int[][] mat;
```

The class has the following method, `alter`.

```
public void alter(int c)
{
    for (int i = 0; i < mat.length; i++)
        for (int j = c + 1; j < mat[0].length; j++)
            mat[i][j-1] = mat[i][j];
}
```

If a 3×4 matrix `mat` is

```
1 3 5 7
2 4 6 8
3 5 7 9
```

then `alter(1)` will change `mat` to

(A)
```
1 5 7 7
2 6 8 8
3 7 9 9
```

then `alter(1)` will change `mat` to

(A)
```
1 5 7 7
2 6 8 8
3 7 9 9
```

(B)
```
1 5 7
2 6 8
3 7 9
```

(C)
```
1 3 5 7
3 5 7 9
```

(D)
```
1 3 5 7
3 5 7 9
3 5 7 9
```

(E)
```
1 7 7 7
2 8 8 8
3 9 9 9
```

17. Refer to the following method:

```
/* Deletes maximum item, i.e., item of lowest priority, from
 * PriorityQueue pq.
 * Precondition:  pq is nonempty.
 * Postcondition: Returns a PriorityQueue that contains the same
 *                elements as pq except for the maximum element,
 *                which has been removed.  */
public PriorityQueue<Type> deleteMax(PriorityQueue<Type> pq)
{
    /* code to delete maximum item */
    return priQ;
}
```

Which of the following replacements for `/* code to delete maximum item */` satisfy the postcondition for the method?

I
```
PriorityQueue<Type> priQ = new PriorityQueue<Type>();
pq.remove();
priQ = pq;
```

II
```
Stack<Type> s = new Stack<Type>();
while (!pq.isEmpty())
    s.push(pq.remove());
```

II
```
Stack<Type> s = new Stack<Type>();
while (!pq.isEmpty())
    s.push(pq.remove());
s.pop();
PriorityQueue<Type> priQ = new PriorityQueue<Type>();
while (!s.isEmpty())
    priQ.add(s.pop());
```

III
```
Queue<Type> q = new LinkedList<Type>();
while (!pq.isEmpty())
    q.add(pq.remove());
q.remove();
PriorityQueue<Type> priQ = new PriorityQueue<Type>();
while (!q.isEmpty())
    priQ.add(q.remove());
```

(A) I only
(B) II only
(C) III only
(D) I and II only
(E) I and III only