Questions 23–25 are based on the following procedure, which copies items from an array `arr` containing `n` distinct numbers into a binary search tree `tree` and then prints the elements.

Procedure:

Step 1: Initialize `tree` to be empty.
Step 2: Insert `arr[0]`, `arr[1]`, . . . , `arr[n-1]` into `tree` using a standard algorithm for insertion of item `arr[i]` into `tree`. (Assume that the insert operation does no balancing of `tree`.)
Step 3: Print the elements stored in `tree`, using an inorder traversal.

23. Which of the following best characterizes the output produced in Step 3 of the above procedure?
    (A) The items are printed in the original order in which they appear in array `arr`.
    (B) The items are printed in sorted order, from smallest to largest.
    (C) The items are printed in sorted order, from largest to smallest.
    (D) The items are printed in the reverse of the order in which they appear in array `arr`.
    (E) The items are printed in random order.

24. Which best describes the best case run time of the whole procedure?

(A) $O(1)$
(B) $O(n)$
(C) $O(\log n)$
(D) $O(n \log n)$
(E) $O(n^2)$

25. The procedure is most likely to exhibit its best case run time when the numbers are stored in array `arr` in which of the following ways?

I Ascending order
II Descending order
III Random order

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

Questions 26 and 27 are based on the `Computable` interface and `LargeInt` class shown below.

```
public interface Computable
{
    Object add(Object obj);          //returns this object + obj
    Object subtract(Object obj);     //returns this object - obj
    Object multiply(Object obj);     //returns this object * obj
}

public class LargeInt implements Comparable, Computable
{
    //private instance variables
        ...

    public LargeInt(int n)      //converts n to LargeInt
    {
        ...
    }

    public String toString()    //returns this LargeInt as a String
    {
        ...
    }
```

```
    Object multiply(Object obj);    //returns this object * obj
}

public class LargeInt implements Comparable, Computable
{
    //private instance variables
        ...

    public LargeInt(int n)     //converts n to LargeInt
    {
        ...
    }

    public String toString()   //returns this LargeInt as a String
    {
        ...
    }

    public Object add(Object obj)  //returns this LargeInt + obj
                                   //precondition: obj of type LargeInt
    {
        ...
    }

    public Object subtract(Object obj)  //returns this LargeInt - obj
                                //precondition: obj of type LargeInt
    {
```

```
    public Object add(Object obj)  //returns this LargeInt + obj
                                   //precondition: obj of type LargeInt
    {
        ...
    }

    public Object subtract(Object obj)  //returns this LargeInt - obj
                                //precondition: obj of type LargeInt
    {
        ...
    }

    public Object multiply(Object obj)  //returns this LargeInt * obj
                                //precondition: obj of type LargeInt
    {
        ...
    }

    //Returns -1 if this LargeInt is less than obj, 1 if it is greater
    // than obj, and 0 if it equals obj.
    public int compareTo(Object obj)
    {
        ...
    }
}
```

26. Of the following pairs of methods, which should be coded and tested first to facilitate testing and debugging the other methods?
    (A) The constructor and `add` method
    (B) The constructor and `compareTo` method
    (C) The constructor and `toString` method
    (D) The `toString` and `compareTo` methods
    (E) The `toString` and one of the `add`, `subtract`, or `multiply` methods

27. Consider the problem of simulating the following loop for `LargeInt` objects:

```
for (int i = 1; i < n; i++)
    System.out.println(i);
```

The following code is used. You may assume that `n` exists and is of type `LargeInt`.

```
LargeInt i = new LargeInt(1);
LargeInt one = new LargeInt(1);
while (i.compareTo(n) < 0)
{
    System.out.println(i);
```

27. Consider the problem of simulating the following loop for `LargeInt` objects:

```
for (int i = 1; i < n; i++)
    System.out.println(i);
```

The following code is used. You may assume that `n` exists and is of type `LargeInt`.

```
LargeInt i = new LargeInt(1);
LargeInt one = new LargeInt(1);
while (i.compareTo(n) < 0)
{
    System.out.println(i);
    /* statement */
}
```

Which of the following should replace `/* statement */` to simulate the loop correctly?

(A) `i = (LargeInt) i.add(one);`
(B) `i = (LargeInt) i.add(1);`
(C) `i = (LargeInt) one.add(n);`
(D) `i = (LargeInt) n.add(one);`
(E) `i = (LargeInt) i.add(n);`

(C) `i = (LargeInt) one.add(n);`
(D) `i = (LargeInt) n.add(one);`
(E) `i = (LargeInt) i.add(n);`

28. A teacher needs to assess the reading level of a textbook. One measure used is the frequency of words that have six or more letters. A computer program scans the text and keeps track of such words and their corresponding frequencies by storing them in a `TreeMap` data structure.

Assuming that there are n different words in the key set so far, which is a *true* statement about operations performed on this frequency map?

(A) Insertion of a new word into the map is $O(1)$.
(B) To check whether a given word is in the map is $O(\log n)$.
(C) To update the frequency of an existing word in the map is $O(1)$.
(D) To print a list of the `keySet` of words in alphabetical order is $O(\log n)$.
(E) To print a list of all word/frequency pairs is $O(\log n)$.

29. A set `set1` is said to be a *subset* of `set2`, `set1` ⊆ `set2`, if there is no element in `set1` that is not in `set2`. Consider the `isSubset` method below:

```
//Precondition:  set1 and set2 are initialized with objects of the
//               same type, SomeType.
//Postcondition: Returns true if set1 is a subset of set2, false
//               otherwise.
public boolean isSubset(Set<SomeType> set1, Set<SomeType> set2)
{
    /* implementation code */
}
```

Which `/* implementation code */` achieves the desired postcondition? Assume the existence of the following method:

```
//Postcondition: Returns a HashSet that contains all the elements
//               of Set s.
public HashSet<SomeType> copySetToHashSet(Set<SomeType> s)
```

I
```
Set<SomeType> temp = copySetToHashSet(set2);
for (SomeType element : set1)
    temp.add(element);
return temp.size() == set2.size();
```

```
//                  of Set s.
public HashSet<SomeType> copySetToHashSet(Set<SomeType> s)
```

I
```
Set<SomeType> temp = copySetToHashSet(set2);
for (SomeType element : set1)
    temp.add(element);
return temp.size() == set2.size();
```

II
```
for (SomeType element : set1)
    if (!set2.contains(element))
        return false;
return true;
```

III
```
for (SomeType element : set1)
{
    for (SomeType e : set2)
        if (!element.equals(e))
            return false;
}
return true;
```

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