12/06/13 APCS – Monta Vista High School ArrayList & Sorting Name_____

1. Describe the action on the ArrayList (or array) elements for the **Merge** SORT.

67	34	83	21	78	44	91	17	6	58	39

2. Describe the action on the ArrayList (or array) elements through the larger **for** loop of the **Insertion** SORT.

I	**AM**	**WELL**	**PREPARED**	**AND**	**WILL**	**ACE**	**THIS**	**TEST**

3. What is the output of the following code segment?

```
ArrayList<String> mylist = new ArrayList<String>();
mylist.add("M");
mylist.add("O");
mylist.add("N");
mylist.add("T");
mylist.add("A");

for (int count = 0; count <= mylist.size()/2; count++)
{
        mylist.remove(count);
}
for (int count = 1; count < mylist.size(); count++)
{
        mylist.add(2,"VISTA");
}
for (String value : mylist)
{
        System.out.print(value + "   ");
}
```

(A) **N T VISTA VISTA A**
(B) **N T VISTA VISTA VISTA A**
(C) **O T VISTA VISTA A**
(D) **O T VISTA VISTA VISTA A**
(E) **An endless loop is created, so nothing prints**

4. Assume that an **ArrayList** of **Integer** values has been declared as follows and has been initialized.

```
ArrayList<Integer> number = new ArrayList<Integer>();
```

Determine which of the following code segments correctly swaps the values contained at indices 0 and 5.

I.

```
Integer temp = number.get(5);
number.remove(0);
number.add(5,number.get(0));
number.remove(5);
number.add(0,temp);
```

II.

```
Integer temp = number.get(5);
number.set(5,number.get(0));
number.set(0,temp);
```

III.

```
Integer temp2 = number.get(0);
Integer temp1 = number.get(5);
number.set(5,temp2);
number.set(0,temp1);
```

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

5. The following incomplete method is intended to sort its array parameter **array** in increasing order.

```
// postcondition: array is sorted in increasing order
public static void sortArray(int [] array)
{
        for (int outer = 1; outer < array.length; outer++)
        {
                int position = outer;
                int key = array[position];

                // Shift larger values to the right
                while (position > 0 && array[position - 1] > key)
                {
                        /* missing code */
                        position--;
                }
                array[position] = key;
        }
}
```

Which of the following could be used to replace **/* missing code */** so that executing the code segment sorts the values in array **array**?

(A) **array[position-1] = array[position];**
(B) **array[position] = array[position - 1];**
(C) **array[position] = array[key];**
(D) **array[key] = array[position];**
(E) **array[key] = array[position - 1];**

6. Consider the following class definitions.

```
public class ClassTwo extends ClassOne
{
        public void methodTwo ( )  {  }
}
public class ClassOne
{
        public void methodOne ( )  {  }
}
```

Now, consider the following declarations in a client class. You may assume that ClassOne and ClassTwo have default constructors.

```
ClassOne c1 = new ClassOne();
ClassTwo c2 = new ClassTwo();
```

Which of the following method calls will cause an error?

I. **c1.methodTwo();** II. **c2.methodTwo();** III. **c2.methodOne();**

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

7. Consider the following three methods that appear within a single class.

```
public void start ( )
{
        ArrayList<Integer> nums = new ArrayList<Integer>(Arrays.asList(1, 1, 2, 3, 5));

        changeIt(nums);
        for(int k = 0; k < nums.size(); k++)
        {
                System.out.print(nums.get(k) .intValue() + " ");
        }

        changeItAgain(nums);
        for(int k = 0; k < nums.size(); k++)
        {
                System.out.print(nums.get(k) .intValue() + "  ");
        }
}

public void changeIt (ArrayList<Integer> list)
{
        list = new ArrayList<Integer>();

        for(int x = 0; x < 5; x++)
        {
                list.add(new Integer(0));
        }
}

public void changeItAgain (ArrayList<Integer> list)
{
        for(int x = 0; x < list.size(); x++)
        {
                list.set(x,new Integer(x));
        }
}
```

What is printed as a result of the call **start()** ?

(A) 0 1 2 3 4 0 1 2 3 4
(B) 0 0 0 0 0 0 0 0 0 0
(C) 0 0 0 0 0 0 1 2 3 4
(D) 1 1 2 3 5 0 1 2 3 4
(E) 1 1 2 3 5 1 1 2 3 5

8. What happens if you forget to initialize a class field that is an object, and start calling its methods?

(A) A syntax error is reported
(B) A run-time "null reference exception" error is reported
(C) A new object is created with the default values for its fields
(D) A new object is created with unpredictable values for its fields
(E) Mr. DeRuiter will tap you on the shoulder and say "bad programmer!" (this is not the correct answer . . .)

9. Which of the following declarations will cause an **error**?

I. ArrayList<String> stringList = new ArrayList<String>();
II. ArrayList<int> intList = new ArrayList<int>();
III. ArrayList<Comparable> compList = new ArrayList<Comparable>();

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

10. The following method is intended to remove from **List<Integer> list** all elements whose value is less than zero:

```
public void removeNegatives (List<Integer> list)
{
        int i = 0, n = list.size();

        while (i < n)
        {
                if (list.get(i) < 0)
                {
                        list.remove(i);
                        n--;
                }
                i++;
        }
}
```

For which list of **Integer** values does this method work as intended?

(A) Only an empty list
(B) All lists that do not contain negative values in consecutive positions
(C) All lists where all the negative values occur before all the positive values
(D) All lists where all the positive values occur before all the negative values
(E) All lists

11. Consider the following code fragment. You may assume that **wordList** has been declared as **ArrayList<String>**.

```
for (String s : wordList)
{
        if (s.length() < 4)
        {
                System.out.println("This word is short.");
        }
}
```

What is the maximum number of times that **"This word is short."** can be printed?
(A) **3**
(B) **4**
(C) **wordList.size()**
(D) **wordList.size() - 1**
(E) **s.length()**

Questions **12** and **13** refer to the following class, **SortOf**:

```
public class SortOf
{
        public static void sort (String [] items)
        {
                int n = items.length;
                while (n > 1)
                {
                        sortHelper(items,n - 1);
                        n--;
                }
        }
        private static void sortHelper (String [] items, int last)
        {
                int m = last;
                for(int k = 0; k < last; k++)
                {
                        if(items[k].compareTo(items[m]) > 0)
                        {
                                m = k;
                        }
                }
                String temp = items[m];
                items[m] = items[last];
                items[last] = temp;
        }
}
```

12. The sorting algorithm implemented in the **sort** method can best be described as:

(A) a **Selection** Sort variation
(B) an **Insertion** Sort variation
(C) a **Bubble** Sort variation
(D) a **Merge** Sort variation
(E) Incorrect implementation of a sorting algorithm

13. Suppose **names** is an array of **String** objects:

```
String [] names = {"Dan", "Alice", "Claire", "Evan", "Boris"};
```

If **SortOf.sort(names)** is running, what is the order of the values in **names** after two complete iterations through the **while** loop in the **sort** method?

(A) **"Boris", "Alice", "Claire", "Dan", "Evan"**
(B) **"Alice", "Claire", "Boris", "Dan", "Evan"**
(C) **"Alice", "Boris", "Claire", "Evan", "Dan"**
(D) **"Alice", "Claire", "Dan", "Evan", "Boris"**
(E) None of the above

14. The number of comparisons in **Bubble** Sort in an array of n elements is roughly proportional to

(A) n (B) n log n (C) $(\log n)^2$ (D) n^2 (E) n^3

15. Consider the following method, **isSorted**, which is intended to return **true** if an array of integers is sorted in nondecreasing order and to return **false** otherwise.

```
public static boolean isSorted (int [] data)
{
        /*  missing code  */
}
```

Which of the following can be used to replace **/* missing code */** so that **isSorted** will work as intended?

I.
```
for (int k = 0; k < data.length; k++)
{
        if (data[k] > data[k + 1])
                return false;
}
return true;
```

II.
```
for (int k = 1; k < data.length; k++)
{
        if (data[k - 1] > data[k])
                return false;
}
return true;
```

III.
```
for (int k = 0; k < data.length - 1; k++)
{
        if (data[k] > data[k + 1])
                return false;
        else
                return true;
}
return true;
```

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

16. Consider the following class:

```
public class ClassList
{
        //  contains the list of student names in a class.
        private ArrayList<String> studentnames;

        //  constructors and other methods and variables not shown.

        public ArrayList<String> getnames ( )
        {
                return studentnames;
        }
}
```

If **ClassList myClass** is declared and initialized in some other client class, which of the following correctly assigns to name the name of the first student in the **myClass** list?

I. `String name = myClass.studentnames[0];`
II. `String name = myClass.studentnames.get(0);`
III. `String name = myClass.getnames().get(0);`

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

17.