Sorting Practice

The following method is a correct implementation of the insertion sort algorithm. The method correctly sorts the elements of numList so that they appear in order from least to greatest.

```
public static void insertionSort(ArrayList<Integer> numList)
{
  for (int j = 1; j < numList.size(); j++)
  {
    int temp = numList.get(j);
    int k = j;
    while (k > 0 && temp < numList.get(k - 1))
    {
      numList.set(k, numList.get(k - 1));
      k--;
    }
    numList.set(k, temp);
    /* end of outer loop */
  }
}
```

Assume that insertionSort has been called with an ArrayList parameter that has been initialized with the following Integer objects.

[60, 30, 10, 20, 50, 40]

Sorting Practice

1. What will the contents of numList be after three passes of the outer loop (i.e., when j == 3 at the point indicated by /* end of outer loop */)?

(A) [10, 20, 30, 50, 60, 40]

(B) [10, 20, 30, 60, 50, 40]

(C) [10, 30, 60, 20, 50, 40]

(D) [20, 10, 30, 60, 50, 40]

Consider the following method. This method correctly sorts the elements of array data into increasing order.

```
public static void sort(int[] data)
{
  for (int j = 0; j < data.length - 1; j++)
  {
    int m = j;
    for (int k = j + 1; k < data.length; k++)
    {
      if (data[k] < data[m])    /* Compare values */
      {
        m = k;
      }
    }
    int temp = data[m];    /* Assign to temp */
    data[m] = data[j];
    data[j] = temp;
    /* End of outer loop */
  }
}
```

Sorting Practice

2. Assume that sort is called with the array {6, 3, 2, 5, 4, 1}. Which of the following represents the contents of data after three passes of the outer loop (i.e., when j == 2 at the point indicated by /* End of outer loop */)?

(A) {1, 2, 3, 4, 5, 6}

(B) {1, 2, 3, 5, 4, 6}

(C) {1, 2, 3, 6, 5, 4}

(D) {1, 3, 2, 5, 4, 6}

Sorting Practice

3. Assume that sort is called with the array {1, 2, 3, 4, 5, 6}. How many times will the expression indicated by /* Compare values */ and the statement indicated by /* Assign to temp */ execute?

(A)

/* Compare values */	/* Assign to temp */
15 times	5 times

(B)

/* Compare values */	/* Assign to temp */
15 times	6 times

(C)

/* Compare values */	/* Assign to temp */
21 times	5 times

(D)

/* Compare values */	/* Assign to temp */
21 times	6 times

Sorting Practice

The following method is a correct implementation of the selection sort algorithm. The method sorts the elements of arr so that they are in order from least to greatest.

```
public static void selectionSort(int[] arr)
{
  for (int j = 0; j < arr.length - 1; j++)
  {
    int minIndex = j;
    for (int k = j + 1; k < arr.length; k++)
    {
      if (arr[k] < arr[minIndex])
      {
        minIndex = k;
      }
    }

    int temp = arr[j];
    arr[j] = arr[minIndex];
    arr[minIndex] = temp;
    /* end of outer loop */
  }
}
```

Assume that selectionSort has been called with an int[] argument that has been initialized with the following contents.

{40, 30, 50, 60, 10, 20}

Sorting Practice

4. What will the contents of `arr` be after three iterations of the outer loop (i.e., when `j == 2` at the point indicated by `/* end of outer loop */`)?

(A) {10, 20, 30, 40, 60, 50}

(B) {10, 20, 30, 60, 40, 50}

(C) {10, 20, 50, 60, 40, 30}

(D) {10, 30, 50, 60, 40, 20}

Sorting Practice

Consider the following correct implementation of the selection sort algorithm.

```
public static void selectionSort(int[] elements)
{
  for (int j = 0; j < elements.length - 1; j++)
  {
    int minIndex = j;
    for (int k = j + 1; k < elements.length; k++)
    {
      if (elements[k] < elements[minIndex])
      {
        minIndex = k;
      }
    }
    if (j != minIndex)
    {
      int temp = elements[j];
      elements[j] = elements[minIndex];
      elements[minIndex] = temp;        // Line 17
    }
  }
}
```

The following declaration and method call appear in a method in the same class as selectionSort.

```
int[] arr = {9, 8, 7, 6, 5};
selectionSort(arr);
```

5. How many times is the statement in line 17 of the method executed as a result of the call selectionSort(arr)?

(A) 2

(B) 3

(C) 4

(D) 5

Sorting Practice

The following method is intended to sort an array of integers into ascending order.

```
public static void sort(int[] arr)
{
  for (int j = 0; j < arr.length - 1; j++)
  {
    int minIndex = j;
    for (int k = j + 1; k < arr.length - 1; k++)
    {
      if (arr[k] < arr[minIndex])
      {
        minIndex = k;
      }
    }
    int temp = arr[minIndex];
    arr[minIndex] = arr[j];
    arr[j] = temp;
  }
}
```

6. This method works as intended for some, but not all, int arrays. Which of the following arrays can be used to show that the method does not work as intended?

(A) {1, 2, 3, 4, 5}

(B) {1, 2, 3, 5, 4}

(C) {1, 2, 4, 3, 5}

(D) {1, 3, 2, 4, 5}

Sorting Practice

The following method is a correct implementation of the insertion sort algorithm. The method correctly sorts the elements of arr so that they appear in order from least to greatest.

```
public static void insertionSort(int[] arr)
{
  for (int j = 1; j < arr.length; j++)
  {
    int temp = arr[j];
    int possibleIndex = j;
    while (possibleIndex > 0 && temp < arr[possibleIndex - 1])
    {
      arr[possibleIndex] = arr[possibleIndex - 1];
      possibleIndex--;                    // Line 10
    }
    arr[possibleIndex] = temp;
  }
}
```

The following code segment appears in a method in the same class as insertionSort.

```
int[] numbers = {4, 12, 4, 7, 19, 6};
insertionSort(numbers);
```

7. How many times is the statement in line 10 of the method executed as a result of the call to insertionSort?

(A) 2

(B) 3

(C) 4

(D) 5

Sorting Practice

The following method is a correct implementation of the selection sort algorithm. The method correctly sorts the elements of data so that they appear in order from least to greatest.

```
public static void selectionSort(int[] data)
{
  for (int j = 0; j < data.length - 1; j++)
  {
    int minIndex = j;

    for (int k = j + 1; k < data.length; k++)
    {
      if (data[k] < data[minIndex])
      {
        minIndex = k;     // Line 11
      }
    }

    if (j != minIndex)
    {
      int temp = data[j];
      data[j] = data[minIndex];
      data[minIndex] = temp;
    }
  }
}
```

The following code segment appears in a method in the same class as selectionSort.

```
int[] vals = {5, 10, 2, 1, 12};
selectionSort(vals);
```

8. How many times is the statement in line 11 of the method executed as a result of the call to selectionSort?

(A) 1

(B) 2

(C) 3

(D) 4

Sorting Practice

The following method is a correct implementation of the selection sort algorithm. The method correctly sorts the elements of arr so that they appear in order from least to greatest.

```
public static void selectionSort(int[] arr)
{
  for (int j = 0; j < arr.length - 1; j++)
  {
    int minIndex = j;
    for (int k = j + 1; k < arr.length; k++)
    {
      if (arr[k] < arr[minIndex])
      {
        minIndex = k;
      }
    }
    int temp = arr[j];
    arr[j] = arr[minIndex];
    arr[minIndex] = temp;
    /* end of outer loop */
  }
}
```

9. Which of the following could be the contents of arr after two passes of the outer loop (i.e., when `j == 1` at the point indicated by `/* end of outer loop */`)?

(A) {10, 20, 50, 40, 30}

(B) {10, 30, 50, 40, 20}

(C) {20, 10, 50, 40, 30}

(D) {20, 30, 50, 40, 10}

Sorting Practice

10. Consider the static method selectSort shown below. Method selectSort is intended to sort an array into increasing order; however, it does not always work as intended.

```
// precondition:  numbers.length > 0

// postcondition: numbers is sorted in increasing order

public static void selectSort(int[] numbers)

{

  int temp;

Line 1:   for (int j = 0; j < numbers.length - 1; j++)

   {

Line 2:    int pos = 0;

Line 3:    for (int k = j + 1; k < numbers.length; k++)

                {

Line 4:     if (numbers[k] < numbers[pos])

                    {

Line 5:      pos = k;

                     }

                }

   temp = numbers[j];

   numbers[j] = numbers[pos];

   numbers[pos] = temp;

  }

}
```

Which of the following changes should be made so that selectSort will work as intended?

Sorting Practice

(A) Line 1 should be changed to
```
for (int j = 0; j < numbers.length - 2; j++)
```

(B) Line 2 should be changed to
```
int pos = j;
```

(C) Line 3 should be changed to
```
for (int k = 0; k < numbers.length; k++)
```

(D) Line 4 should be changed to
```
if (numbers[k] > numbers[pos])
```

Sorting Practice

11. The following incomplete method is intended to sort its array parameter arr in increasing order.

```
// postcondition: arr is sorted in increasing order
public static void sortArray(int[] arr)
{
  int j, k;

  for (j = arr.length - 1; j > 0; j--)
  {
    int pos = j;

    for ( /* missing code */ )
    {
      if (arr[k] > arr[pos])
      {
        pos = k;
      }
    }
    swap(arr, j, pos);
  }
}
```

Assume that swap(arr, j, pos) exchanges the values of arr[j] and arr[pos]. Which of the following could be used to replace /* *missing code* */ so that executing the code segment sorts the values in array arr?

Sorting Practice

(A) `k = j - 1; k > 0; k--`

(B) `k = j - 1; k >= 0; k--`

(C) `k = 1; k < arr.length; k++`

(D) `k = 0; k <= arr.length; k++`

Consider the following correct implementation of the selection sort algorithm.

```
public static void selectionSort(int[] elements)
{
  for (int j = 0; j < elements.length - 1; j++)
  {
    int minIndex = j;
    for (int k = j + 1; k < elements.length; k++)
    {
      if (elements[k] < elements[minIndex])
      {
        minIndex = k;
      }
    }
    if (j != minIndex)
    {
      int temp = elements[j];
      elements[j] = elements[minIndex];
      elements[minIndex] = temp;   // line 17
    }
  }
}
```

The following declaration and method call appear in a method in the same class as selectionSort.

```
int[] arr = {30, 40, 10, 50, 20};
selectionSort(arr);
```

12. How many times is the statement `elements[minIndex] = temp;` in line 17 of the method executed as a result of the call to selectionSort?

(A) 2

(B) 3

(C) 4

(D) 5

Sorting Practice

13. Consider the following `mergeSortHelper` method, which is part of an algorithm to recursively sort an array of integers.

```
/** Precondition: (arr.length == 0 or 0 <= from <= to <= arr.length)
 * arr.length == temp.length
 */
public static void mergeSortHelper(int[] arr,
   int from, int to, int[] temp)
{
     if (from < to)
     {
         int middle = (from + to) / 2;
         mergeSortHelper(arr, from, middle, temp);
         mergeSortHelper(arr, middle + 1, to, temp);
         merge(arr, from, middle, to, temp);
     }
}
```

The `merge` method is used to merge two halves of an array (`arr[from]` through `arr[middle]`, inclusive, and `arr[middle + 1]` through `arr[to]`, inclusive) when each half has already been sorted into ascending order. For example, consider the array `arr1`, which contains the values `{1, 3, 5, 7, 2, 4, 6, 8}`. The lower half of `arr1` is sorted in ascending order (elements `arr1[0]` through `arr1[3]`, or `{1, 3, 5, 7}`), as is the upper half of `arr1` (elements `arr1[4]` through `arr1[7]`, or `{2, 4, 6, 8}`). The array will contain the values `{1, 2, 3, 4, 5, 6, 7, 8}` after the method call `merge(arr1, 0, 3, 7, temp)`. The array `temp` is a temporary array declared in the calling program.

Consider the following code segment, which appears in a method in the same class as `mergeSortHelper` and `merge`.

```
int[] vals = {80, 50, 30, 20, 60, 70};
int[] temp = new int[vals.length];
mergeSortHelper(vals, 0, vals.length - 1, temp);
```

Which of the following represents the arrays merged the last time the `merge` method is executed as a result of the code segment above?

(A) `{20, 30, 50}` and `{60, 70, 80}` are merged to form `{20, 30, 50, 60, 70, 80}`.

(B) `{20, 50, 70}` and `{30, 60, 80}` are merged to form `{20, 30, 50, 60, 70, 80}`.

(C) `{20, 50, 70}` and `{30, 60, 80}` are merged to form `{20, 50, 70, 30, 60, 80}`.

(D) `{30, 50, 80}` and `{20, 60, 70}` are merged to form `{20, 30, 50, 60, 70, 80}`.

Sorting Practice

Consider the following mergeSortHelper method, which is part of a correctly implemented algorithm to recursively sort an array of integers.

```
/** Preconditions: (arr.length == 0 or
 *                  0 <= from <= to <= arr.length)
 *                  arr.length == temp.length
 */
public static void mergeSortHelper(int[] arr, int from,
                                   int to, int[] temp)
{
  if (from < to)
  {
    int middle = (from + to) / 2;
    mergeSortHelper(arr, from, middle, temp);
    mergeSortHelper(arr, middle + 1, to, temp);
    merge(arr, from, middle, to, temp);
  }
}
```

The merge method (not shown) is used to merge two halves of an array (arr[from] through arr[middle], inclusive, and arr[middle + 1] through arr[to], inclusive) when each half has already been sorted into ascending order. For example, if arr1 contains {1, 3, 5, 7, 2, 4, 6, 8}, then arr1 will contain {1, 2, 3, 4, 5, 6, 7, 8} after the method call merge(arr1, 0, 3, 7, temp).

Consider the following code segment, which appears in a method in the same class as mergeSortHelper and merge.

```
int[] numbers = {40, 10, 20, 30};
int[] temp = new int[numbers.length];
mergeSortHelper(numbers, 0, numbers.length - 1, temp);
```

14. How many times will the merge method be called as a result of running the code segment?

(A) 1
(B) 2
(C) 3
(D) 4