MULTIPLE-CHOICE QUESTIONS ON SORTING AND SEARCHING

1. The decision to choose a particular sorting algorithm should be made based on

   I Run-time efficiency of the sort
   II Size of the array
   III Space efficiency of the algorithm

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

2. The following code fragment does a sequential search to determine whether a given integer, `value`, is stored in an array `a[0] ... a[n-1]`.

```
int i = 0;
while (/* boolean expression */)
{
    i++;
}
if (i == n)
    return -1;      //value not found
else
    return i;       // value found at location i
```

   Which of the following should replace `/* boolean expression */` so that the algorithm works as intended?
   (A) `value != a[i]`
   (B) `i < n && value == a[i]`
   (C) `value != a[i] && i < n`
   (D) `i < n && value != a[i]`
   (E) `i < n || value != a[i]`

3. A feature of data that is used for a binary search but not necessarily used for a sequential search is
   (A) length of list.
   (B) type of data.
   (C) order of data.
   (D) smallest value in the list.
   (E) median value of the data.

4. Array `unsortedArr` contains an unsorted list of integers. Array `sortedArr` contains a sorted list of integers. Which of the following operations is more efficient for `sortedArr` than `unsortedArr`? Assume the most efficient algorithms are used.

   I Inserting a new element
   II Searching for a given element
   III Computing the mean of the elements

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

5. An algorithm for searching a large sorted array for a specific value x compares every third item in the array to x until it finds one that is greater than or equal to x. When a larger value is found, the algorithm compares x to the previous two items. If the array is sorted in increasing order, which of the following describes all cases when this algorithm uses fewer comparisons to find x than would a binary search?
   (A) It will never use fewer comparisons.
   (B) When x is in the middle position of the array
   (C) When x is very close to the beginning of the array
   (D) When x is very close to the end of the array
   (E) When x is not in the array

6. Assume that `a[0]` ... `a[N-1]` is an array of N positive integers and that the following assertion is true:

   `a[0] > a[k]` for all k such that $0 < k < N$

   Which of the following *must* be true?
   (A) The array is sorted in ascending order.
   (B) The array is sorted in descending order.
   (C) All values in the array are different.
   (D) `a[0]` holds the smallest value in the array.
   (E) `a[0]` holds the largest value in the array.

7. The following code is designed to set `index` to the location of the first occurrence of `key` in array `a` and to set `index` to -1 if `key` is not in `a`.

```
index = 0;
while (a[index] != key)
    index++;
if (a[index] != key)
    index = -1;
```

   In which case will this program *definitely* fail to perform the task described?
   (A) When `key` is the first element of the array
   (B) When `key` is the last element of the array
   (C) When `key` is not in the array
   (D) When `key` equals 0
   (E) When `key` equals `a[key]`

8. Refer to method search.

```
/* Precondition:  v[0]...v[v.length-1] are initialized.
 * Postcondition: Returns k such that -1 <= k <= v.length-1.
 *                If k >= 0 then v[k] == key. If k == -1,
 *                then key != any of the elements in v.  */
public static int search(int[] v, int key)
{
    int index = 0;
    while (index < v.length && v[index] < key)
        index++;
    if (index != v.length)
        return index;
    else
        return -1;
}
```

Assuming that the method works as intended, which of the following should be added to the precondition of search?
(A) v is sorted smallest to largest.
(B) v is sorted largest to smallest.
(C) v is unsorted.
(D) There is at least one occurrence of key in v.
(E) key occurs no more than once in v.

Questions 9–13 are based on the binSearch method and the private instance variable a for some class:

```
private int[] a;

/* Does binary search for key in array a[0]...a[a.length-1],
 *    sorted in ascending order.
 * Postcondition: Returns index such that a[index]==key.
 *                If key not in a, returns -1.  */
public int binSearch(int key)
{
    int low = 0;
    int high = a.length - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (a[mid] == key)
            return mid;
        else if (a[mid] < key)
            low = mid + 1;
        else
            high = mid - 1;
    }
    return -1;
}
```

A binary search will be performed on the following list.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
4	7	9	11	20	24	30	41

9. To find the key value 27, the search interval *after* the first pass through the `while` loop will be
   (A) `a[0]...a[7]`
   (B) `a[5]...a[6]`
   (C) `a[4]...a[7]`
   (D) `a[2]...a[6]`
   (E) `a[6]...a[7]`

10. How many iterations will be required to determine that 27 is not in the list?
   (A) 1
   (B) 3
   (C) 8
   (D) 27
   (E) An infinite loop since 27 is not found

11. What will be stored in y after executing the following?

```
int y = binSearch(4);
```

   (A) 20
   (B) 7
   (C) 4
   (D) 0
   (E) -1

12. If the test for the `while` loop is changed to

```
while (low < high)
```

   the `binSearch` method does not work as intended. Which value in the given list will not be found?
   (A) 4
   (B) 7
   (C) 11
   (D) 24
   (E) 30

13. For `binSearch`, which of the following assertions will be true following every iteration of the `while` loop?
   (A) `key` = `a[mid]` or `key` is not in `a`.
   (B) `a[low]` ≤ `key` ≤ `a[high]`
   (C) `low` ≤ `mid` ≤ `high`
   (D) `key` = `a[mid]`, or `a[low]` ≤ `key` ≤ `a[high]`
   (E) `key` = `a[mid]`, or `a[low]` ≤ `key` ≤ `a[high]`, or `key` is not in array `a`.

14. A large sorted array containing about 30,000 elements is to be searched for a value `key` using an iterative binary search algorithm. Assuming that `key` is in the array, which of the following is closest to the smallest number of iterations that will guarantee that `key` is found? Note: $10^3 \approx 2^{10}$.
    (A) 15
    (B) 30
    (C) 100
    (D) 300
    (E) 3000

For Questions 15–18 refer to the `insertionSort` method and the private instance variable `a`, both in a `Sorter` class.

```
private Comparable[] a;

/* Precondition:  a[0],a[1]...a[a.length-1] is an unsorted array
 *                of Comparable objects.
 * Postcondition: Array a is sorted in descending order.  */
public void insertionSort()
{
    for (int i = 1; i < a.length; i++)
    {
        Comparable temp = a[i];
        int j = i - 1;
        while (j >= 0 && temp.compareTo(a[j]) > 0)
        {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = temp;
    }
}
```

15. An array of `Integer` is to be sorted biggest to smallest using the `insertionSort` method. If the array originally contains

    1 7 9 5 4 12

    what will it look like after the third pass of the `for` loop?
    (A) 9 7 1 5 4 12
    (B) 9 7 5 1 4 12
    (C) 12 9 7 1 5 4
    (D) 12 9 7 5 4 1
    (E) 9 7 12 5 4 1

16. When sorted biggest to smallest with `insertionSort`, which list will need the fewest changes of position for individual elements?
    (A) 5, 1, 2, 3, 4, 9
    (B) 9, 5, 1, 4, 3, 2
    (C) 9, 4, 2, 5, 1, 3
    (D) 9, 3, 5, 1, 4, 2
    (E) 3, 2, 1, 9, 5, 4

17. When sorted biggest to smallest with `insertionSort`, which list will need the greatest number of changes in position?
    (A) 5, 1, 2, 3, 4, 7, 6, 9
    (B) 9, 5, 1, 4, 3, 2, 1, 0
    (C) 9, 4, 6, 2, 1, 5, 1, 3
    (D) 9, 6, 9, 5, 6, 7, 2, 0
    (E) 3, 2, 1, 0, 9, 6, 5, 4

18. While typing the `insertionSort` method, a programmer by mistake enters

```
while (temp.compareTo( a[j]) > 0)
```

instead of

```
while (j >= 0 && temp.compareTo( a[j]) > 0)
```

Despite this mistake, the method works as intended the first time the programmer enters an array to be sorted in descending order. Which of the following could explain this?

I The first element in the array was the largest element in the array.
II The array was already sorted in descending order.
III The first element was less than or equal to all the other elements in the array.

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

19. Consider the following class.

```
/* A class that sorts an array of objects from
 * largest to smallest using a selection sort. */
public class Sorter
{
    private Comparable[] a;

    public Sorter(Comparable[] arr)
    { a = arr; }

    /* Swap a[i] and a[j] in array a. */
    private void swap(int i, int j)
    { /* implementation not shown */ }

    /* Sort array a from largest to smallest using selection sort.
     * Precondition: a is an array of Comparable objects. */
    public void selectionSort()
    {
        for (int i = 0; i < a.length - 1; i++)
        {
            //find max element in a[i+1] to a[n-1]
            Comparable max = a[i];
            int maxPos = i;
            for (int j = i + 1; j < a.length; j++)
                if (max.compareTo(a[j]) < 0) //max less than a[j]
                {
                    max = a[j];
                    maxPos = j;
                }
            swap(i, maxPos);    //swap a[i] and a[maxPos]
        }
    }
}
```

If an array of `Integer` contains the following elements, what would the array look like after the third pass of `selectionSort`, sorting from high to low?

89 42 −3 13 109 70 2

(A) 109 89 70 13 42 −3 2
(B) 109 89 70 42 13 2 −3
(C) 109 89 70 −3 2 13 42
(D) 89 42 13 −3 109 70 2
(E) 109 89 42 −3 13 70 2

20. The elements in a long list of integers are roughly sorted in decreasing order. No more than 5 percent of the elements are out of order. Which of the following is a valid reason for using an insertion sort rather than a selection sort to sort this list into decreasing order?

    I There will be fewer comparisons of elements for insertion sort.
    II There will be fewer changes of position of elements for insertion sort.
    III There will be less space required for insertion sort.

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

Optional topic

21. The code shown sorts array `a[0] ...a[a.length-1]` in descending order.

```
public static void sort(Comparable[] a)
{
    for (int i = 0; i < a.length - 1; i++)
        for (int j = 0; j < a.length - i - 1; j++)
            if (a[j].compareTo(a[j+1]) < 0)
                swap(a, j, j + 1);  //swap a[j] and a[j+1]
}
```

    This is an example of
    (A) selection sort.
    (B) insertion sort.
    (C) mergesort.
    (D) quicksort.
    (E) none of the above.

22. Which of the following is a valid reason why mergesort is a better sorting algorithm than insertion sort for sorting long lists?

    I Mergesort requires less code than insertion sort.
    II Mergesort requires less storage space than insertion sort.
    III Mergesort runs faster than insertion sort.

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

23. A large array of lowercase characters is to be searched for the pattern "pqrs." The first step in a very efficient searching algorithm is to look at characters with index
    (A) 0, 1, 2, ... until a "p" is encountered.
    (B) 0, 1, 2, ... until any letter in "p" ... "s" is encountered.
    (C) 3, 7, 11, ... until an "s" is encountered.
    (D) 3, 7, 11, ... until any letter in "p" ... "s" is encountered.
    (E) 3, 7, 11, ... until any letter other than "p" ... "s" is encountered.

24. The array names[0], names[1], …, names[9999] is a list of 10,000 name strings. The list is to be searched to determine the location of some name X in the list. Which of the following preconditions is necessary for a binary search?
    (A) There are no duplicate names in the list.
    (B) The number of names N in the list is large.
    (C) The list is in alphabetical order.
    (D) Name X is definitely in the list.
    (E) Name X occurs near the middle of the list.

25. Consider the following method:

```
//Precondition: a[0],a[1]...a[n-1] contain integers.
public static int someMethod(int[] a, int n, int value)
{
    if (n == 0)
        return -1;
    else
    {
        if (a[n-1] == value)
            return n - 1;
        else
            return someMethod(a, n - 1, value);
    }
}
```

The method shown is an example of
    (A) insertion sort.
    (B) mergesort.
    (C) selection sort.
    (D) binary search.
    (E) sequential search.

Optional topic

26. The partition method for quicksort partitions a list as follows:
    (i) A pivot element is selected from the array.
    (ii) The elements of the list are rearranged such that all elements to the left of the pivot are less than or equal to it; all elements to the right of the pivot are greater than or equal to it.

Partitioning the array requires which of the following?
    (A) A recursive algorithm
    (B) A temporary array
    (C) An external file for the array
    (D) A swap algorithm for interchanging array elements
    (E) A merge method for merging two sorted lists

27. Assume that mergesort will be used to sort an array arr of n integers into increasing order. What is the purpose of the merge method in the mergesort algorithm?
    (A) Partition arr into two parts of roughly equal length, then merge these parts.
    (B) Use a recursive algorithm to sort arr into increasing order.
    (C) Divide arr into n subarrays, each with one element.
    (D) Merge two sorted parts of arr into a single sorted array.
    (E) Merge two sorted arrays into a temporary array that is sorted.

28. A binary search is to be performed on an array with 600 elements. In the *worst* case, which of the following best approximates the number of iterations of the algorithm?
    (A) 6
    (B) 10
    (C) 100
    (D) 300
    (E) 600

29. A worst case situation for insertion sort would be

    I A list in correct sorted order.
    II A list sorted in reverse order.
    III A list in random order.

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

Optional topic

Questions 30 and 31 are based on the Sort interface and MergeSort and QuickSort classes shown below.

```
public interface Sort
{
    void sort();
}

public class MergeSort implements Sort
{
    private Comparable[] a;

    //constructor
    public MergeSort(Comparable[] arr)
    { a = arr; }

    //Merge a[lb] to a[mi] and a[mi+1] to a[ub].
    //Precondition: a[lb] to a[mi] and a[mi+1] to a[ub] both
    //              sorted in increasing order.
    private void merge(int lb, int mi, int ub)
    { /* Implementation not shown. */ }

    //Sort a[first]..a[last] in increasing order using mergesort.
    //Precondition: a is an array of Comparable objects.
    private void sort(int first, int last)
    {
        int mid;

        if (first != last)
        {
            mid = (first + last) / 2;
            sort(first, mid);
            sort(mid + 1, last);
            merge(first, mid, last);
        }
    }

    //Sort array a from smallest to largest using mergesort.
    //Precondition: a is an array of Comparable objects.
    public void sort()
    {
        sort(0, a.length - 1);
    }
}

public class QuickSort implements Sort
{
    private Comparable[] a;

    //constructor
    public QuickSort(Comparable[] arr)
    { a = arr; }

    //Swap a[i] and a[j] in array a.
    private void swap(int i, int j)
    { /* Implementation not shown. */ }
```

(continued)

```
    //Returns the index pivPos such that a[first] to a[last]
    //is partitioned.
    //a[first..pivPos] <= a[pivPos] and a[pivPos..last] >= a[pivPos]
    private int partition(int first, int last)
    { /* Implementation not shown. */ }

    //Sort a[first]..a[last] in increasing order using quicksort.
    //Precondition: a is an array of Comparable objects.
    private void sort(int first, int last)
    {
        if (first < last)
        {
            int pivPos = partition(first, last);
            sort(first, pivPos - 1);
            sort(pivPos + 1, last);
        }
    }

    //Sort array a in increasing order.
    public void sort()
    {
        sort(0, a.length - 1);
    }
}
```

30. Notice that the `MergeSort` and `QuickSort` classes both have a private helper method that implements the recursive sort routine. For this example, which of the following is *not* a valid reason for having a helper method?

    I The helper method hides the implementation details of the sorting algorithm from the user.
    II A method with additional parameters is needed to implement the recursion.
    III Providing a helper method increases the run-time efficiency of the sorting algorithm.

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

(continued)

31. A piece of code to test the `QuickSort` and `MergeSort` classes is as follows:

```
//Create an array of Comparable values
Comparable[] strArray = makeArray(strList);
writeList(strArray);
/* more code */
```

where `makeArray` creates an array of `Comparable` from a list `strList`. Which of the following replacements for `/* more code */` is reasonable code to test `QuickSort` and `MergeSort`? You can assume `writeList` correctly writes out an array of `String`.

(A)
```
Sort q = new QuickSort(strArray);
Sort m = new MergeSort(strArray);
q.sort();
writeList(strArray);
m.sort();
writeList(strArray);
```

(B)
```
QuickSort q = new Sort(strArray);
MergeSort m = new Sort(strArray);
q.sort();
writeList(strArray);
m.sort();
writeList(strArray);
```

(C)
```
Sort q = new QuickSort(strArray);
Sort m = new MergeSort(strArray);
Comparable[] copyArray = makeArray(strList);
q.sort(0, strArray.length - 1);
writeList(strArray);
m.sort(0, copyArray.length - 1);
writeList(copyArray);
```

(D)
```
QuickSort q = new Sort(strArray);
Comparable[] copyArray = makeArray(strList);
MergeSort m = new Sort(strArray);
q.sort();
writeList(strArray);
m.sort();
writeList(copyArray);
```

(E)
```
Sort q = new QuickSort(strArray);
Comparable[] copyArray = makeArray(strList);
Sort m = new MergeSort(copyArray);
q.sort();
writeList(strArray);
m.sort();
writeList(copyArray);
```

32. Consider a binary search algorithm to search an ordered list of numbers. Which of the following choices is closest to the maximum number of times that such an algorithm will execute its main comparison loop when searching a list of 1 million numbers?
    (A) 6
    (B) 20
    (C) 100
    (D) 120
    (E) 1000

33. Consider these three tasks:

    I A sequential search of an array of *n* names
    II A binary search of an array of *n* names in alphabetical order
    III An insertion sort into alphabetical order of an array of *n* names that are initially in random order

    For large *n*, which of the following lists these tasks in order (from least to greatest) of their average case run times?

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