



**1. Matching [ 8 pts ]**

For each algorithm name, select the correct textual description of the algorithm.

1. \_\_\_\_\_ Selection Sort
2. \_\_\_\_\_ Linear Search
3. \_\_\_\_\_ Quick Sort
4. \_\_\_\_\_ Binary Search
  - A.** Adjacent items in the array are compared to each other and swapped as needed until the array is sorted.
  - B.** Selects a pivot element, groups the data relative to the pivot, recurses on each subpart.
  - C.** On a sorted array, an element is found quickly by eliminating half of the array with each comparison.
  - D.** The Y combinator is applied to each element in the array until a least upper bound can be established resulting in a sorted array.
  - E.** The array is scanned sequentially until a desired element is found.
  - F.** The maximum (or minimum) element of the unsorted portion of the array is found, then moved to its final position. This process is repeated until the array is sorted.
  - G.** None of the above.
  - H.** All of the above.

**2. Tracing [ 16 pts ]**

What is the output when the main method in the code below is run?

```
public class Trace2{
    public static void mystery2(int num, char[] data){
        if(num>=data.length){
            System.out.println("Finished");
            return;
        }
        data[num]=Character.toUpperCase(data[num]);
        num++;
        mystery2(num,data);
        System.out.println("num is now "+num);
    }
    public static void main(String[] args){
        char[] data={'a','z','b','c'};
        int num=2;
        mystery2(num,data);
        System.out.println("num= " +num);
        for(int i=0;i<data.length;i++){
            System.out.println(data[i]);
        }
    }
}
```

### 3. Polymorphism [ 12 pts ]

- 4 (a) Explain the concept of polymorphism. Your explanation should make specific reference to
- Inheritance
  - Abstract Classes
  - Interfaces
- 4 (b) Give an example of what polymorphism allows a programmer to do. Be sure to mention
- Collections (arrays, Vectors, etc).
  - Methods
- (you do not have to write code for this problem, but if short code fragments help your explanation, you may include them)
- 4 (c) Explain when/why casting is needed with respect to Objects.

#### 4. Dynamic Binding [ 12 pts ]

Assume that you are given the following classes:

- **Animal** is abstract, and has the abstract method **public boolean isFerocious()**, a method which subclasses implement such that it returns true if the animal is ferocious, false otherwise.
- **Cat** extends **Animal** and implements **isFerocious()** so that it returns false.
- **Lion** extends **Cat** and implements **isFerocious()** so that it returns true.
- **Elephant** extends **Animal** and implements **isFerocious()** so that it returns false.

8

(a) `public int countFerocious(Animal[] zoo)`. This method should be useable with other **Animal** subclasses that may be written later. For this method, you may not use

1. The **instanceof** operator
2. Casting of any sort
3. The `getClass()` method

If you violate the above restriction, you will receive no credit for this question.

Given the following `Animal[]`:

```
Animal[] testZoo={new Cat(), new Lion(), new Lion(), new Elephant()};
```

the **countFerocious** method should return 2.

```
public int countFerocious(Animal[] myZoo) {
```

```
}
```

4

(b) Explain how your method accomplishes its task, with specific reference to the concept of Dynamic Binding.

5. **Arrays [ 16 pts ]**

Write the method **public int[][] vectorToArray(Vector data)** which takes **data**, a Vector whose elements are all Vectors, which in turn contain Integers, and creates a two dimensional array of ints holding the information from **data**. The elements of the returned array should appear in the same order as in the Vector passed in. You may not assume that the output array will be rectangular, it may be "jagged". For example, given [ [ 1, 2] , [ 3, 4, 5] ] your method should return

```
{{1,2},
```

```
{3,4,5}}
```

```
public int[][] vectorToArray(Vector data) {
```

```
}
```

**6. Sorting [ 16 pts ]**

An important part of the merge sort algorithm is the **merge** method. Assume that you have correctly coded all of the merge sort algorithm except for the merge method. Complete the **merge** method below so that it properly merges the given arrays into ascending order.

```
private Comparable[] merge(Comparable[] array1, Comparable [] array2) {
```

```
}
```

## 7. Recursion [ 20 pts ]

Develop the method **stringCount**, that consumes an array of Objects and returns the number of String Objects in the array.

- 5 (a) As a first step, write a helper method called **arrayRest** which takes an array of Objects, and returns a new array containing all items except the first. The returned array will have size 1 less than the array passed in, and should not have any side effects on the array passed in. You may assume that the array passed in is of length at least 1. You are not required to use recursion for this sub-part of the problem (however, if you feel it is useful, you may do so)
- ```
public Object[] arrayRest(Object[] data) {
```

```
}
```

(Question continues on next page ⇒)

- 15 (b) Now, write the **stringCount** method, which (as stated before) counts the number of **Strings** in the array passed in. You may assume that no item in the array is *null*. For this part of the problem, you **must use recursion only**- if you do not use recursion, or if you use any iteration, you will receive no credit. You may assume that the **arrayRest** method which you just wrote works as specified and use it in this problem. Below is an example of what this method should return:

```
Object list[] = { new Double(3),
                  new int[] 1, 2, 3,
                  "Fred",
                  new Object(),
                  "Joe",
                  "Mary" };
stringCount(list) should return 3
public Object[] stringCount(Object[] data) {
```

```
}
```