

1. Matching [8 pts]

For each algorithm name, select **ALL** attributes that correctly describe that algorithm.

1. _____ Merge Sort
2. _____ Binary Search
3. _____ Linear Search
4. _____ Selection Sort
 - A. $O(1)$
 - B. $O(n)$
 - C. $O(n \lg n)$
 - D. $O(n^2)$
 - E. $O(n!)$
 - F. Divide and Conquer
 - G. Requires selection of a pivot element
 - H. Can only be done on a sorted array

2. Tracing [16 pts]

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

```
public class Trace3{
    public static void mystery3(int num, char [] data){
        if(num>=data.length){
            System.out.println("No more to do");
            return;

        }
        data[num]=Character.toLowerCase(data[num]);
        num++;
        mystery3(num,data);
        System.out.println("currently on "+num);
    }
    public static void main(String[] args){
        char[] data={'Q','W','X','M'};
        int num=2;
        mystery3(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 String** , a method which returns the favorite food of the animal.
- **Cat** extends **Animal** and implements so that it returns "Tuna".
- **Lion** extends **Cat** and implements so that it returns "Zebra".
- **Elephant** extends **Animal** and implements so that it returns "Hay".

8

(a) `public String[] foodForEachAnimal(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 foodForEachAnimal method should return {"Tuna","Zebra","Zebra","Hay"}.
public String[] foodForEachAnimal(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 char[][] lettersOfWords(String data)** which takes a String and splits it into tokens, with spaces, tabs, and newlines as the delimiters (the default delimiters for StringTokenizer). Each of these words is then split into its component characters, and an array is formed, where the first index represents the word, and the second the character in the word. For example, if "abc def" were passed in, the resulting array would be

```
{{'a','b','c'}  
{'d','e','f'}}
```

You may find the method **public int countTokens()** in the StringTokenizer class, which returns the number of tokens available to be useful to you.

```
public char[][] lettersOfWords(String data) {
```

```
}
```

6. **Sorting [16 pts]**

Assume that you have the following correctly working methods already implemented:

- `public Comparable[] allSmaller(Comparable[] arr, Comparable target)`
- `public Comparable[] allLarger(Comparable[] arr, Comparable target)`

Write the method `public void quickSort(Comparable[] data)` which sorts the array passed in into descending order. The `allSmaller` method will return all elements of the array passed to it that are smaller than `target`. The `allLarger` method will return all elements of the array passed to it that are larger than `target`. You may assume no duplicate elements occur in the array passed in.

```
public void quickSort(Comparable[] data) {
```

```
}
```

7. Recursion [20 pts]

Given the following Bid class:

```
public class Bid{
    private int bidder;
    private double bid;
    public int getBidder() { return bidder; }
    public double getBid() { return bid; }
    public Bid(int bidder, double bid){
        this.bidder=bidder;
        this.bid=bid;
    }
    public String toString(){
        return "<"+getBidder() +" bids "+ getBid()+">";
    }
}
```

You may treat an array of Bid objects as a list. Develop the method **highestBid**, which consumes a non-empty list of auction bids and returns the winning bid for that item. Assume that at least one bid for the item in question is contained in the list.

- 5 (a) As a first step, write a helper method **bidArrayRest**. This method will take an array of **Bids** and return 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 Bid[] bidArrayRest(Bid[] data) {
```

```
}
```

(Question continues on next page ⇒)

- 15 (b) Now write the **highestBid** which will return the **Bid** object representing the highest bid in the array. 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 **bidArrayrest** method which you just wrote works as specified and use it in this problem. Below is an example of what this method should return:

```
Bid auction[] = { new Bid(3, 2400.),
 new Bid(4, 3200.),
 new Bid(1, 4800.),
 new Bid(5, 8100.),
 new Bid(8, 400.),
 new Bid(2, 800.),
 new Bid(3, 5400.),
 new Bid(7, 8700.),
 new Bid(11,2400.) };
```

highestBid( auction ) should return the Bid <7 bids 8700.0>

```
public Bid highestBid(Bid[] auction) {
```

```
}
```