



**1. Matching[ 10 pts ]**

Choose the **best** word for each statement.

1. \_\_\_\_\_ Provides runtime resolution to the most specific implementation possible.
2. \_\_\_\_\_ Items are taken from an unsorted list, and inserted into a sorted list.
3. \_\_\_\_\_ An abstraction for a sequence of bytes from which data can be read or to which data can be written.
4. \_\_\_\_\_ Has  $O(n \log n)$  in all cases.
5. \_\_\_\_\_ A class like `BufferedReader`, which adds more functionality to an object.
  - A. Hash Table
  - B. Dynamic Binding
  - C. Selection Sort
  - D. Polymorphism
  - E. Merge Sort
  - F. Stream
  - G. Inheritance
  - H. Decorator Class
  - I. Insertion Sort
  - J. Dynamic Binding
  - K. Heap
  - L. Fjord
  - M. Adornment Class

**2. Sorting[ 10 pts ]**

Given the following data items, sort them using the quick sort algorithm in descending(big to small) order. Show the intermediate steps as the data is being sorted.

4	63	-1	2	-5	8	3
---	----	----	---	----	---	---

### 3. Binary Search Trees[ 10 pts ]

- 5 (a) Add the following elements in the order they appear to an empty Binary Search Tree: 7,4,8,2,9,3,1,5,-1,6.

- 5 (b) From the tree you created in part a) remove 1.

#### 4. Hash Tables[10 pts]

Given the following (key,data) pairs, add the items to an empty hash table of **size 5**. Show only the data in the hash table. If a collision occurs, use external chaining to resolve the collision.

(5 , "Homer")

(2 , "Bart")

(0 , "Lisa")

(7 , "Maggie")

(18 , "Marge")

## 5. Exceptions[ 10 pts ]

Given the following class, write the output when the main method is run.

```
public class ExceptionRamaA{
    public ExceptionRamaA() throws NullPointerException{
        throw new NullPointerException();
    }
    public static void main(String[] args){
        System.out.println("Going in");
        try{
            System.out.println("I'm in");
            try{
                System.out.println("Weird");
                ExceptionRamaA first = new ExceptionRamaA();
                System.out.println("OK");
            }
            catch(ArithmeticException ae){
                System.out.println("In here");
            }
        }
        catch(Exception e){
            System.out.println("Confused?");
        }
        finally{ System.out.println("Exeunt"); }
    }
}
```

---

## 6. File I/O[ 10 pts ]

A Misguided Java programmer wrote the following code. With the given input file, he expected the following output, but received something different.

Input File -----	Expected Output -----	Actual Output -----
Somewhere over the rainbow skies are blue	Somewhere over the rainbow skies are blue	

Fix the following code so that the output the programmer receives is the Expected output.

```
import java.io.*;
public class MisguidedIOB{
    public MisguidedIOB(){

        try{
            String line = null;

            BufferedReader br = new BufferedReader(new FileReader("input.txt"));

            while( line != null ){

line = br.readLine();

                System.out.println(line);

            }

            br.close();
        }
        catch(Exception e){System.out.println("Wow");}
    }
    public static void main(String[] args){
        MisguidedIOB myIO = new MisguidedIOB();
    }
}
```

## 7. Coding[ 10 pts ]

Write the method **public String removeBigWords(String line)**, which takes in a String and returns a new String that consists of all the words from the input String that are less than 4 in length. The String class has a `length()` method that returns the length of the String.

## 8. Binary Search Trees[ 10 pts ]

Given the following class `BSTNode`:

```
public class BSTNode{
    private Comparable data;
    private BSTNode right=null, left=null;
    public void setData(Comparable data){ this.data = data; }
    public void setRight(BSTNode right){ this.right = right; }
    public void setleft(BSTNode left){ this.left = left; }
    public Comparable getData(){ return data; }
    public BSTNode getRight(){ return right; }
    public BSTNode getLeft(){ return left; }
    public BSTNode(Comparable data){ this.data = data; }
}
```

Write the `toArray()` method for the `BST` class, which will return an array with the data of the BST. The data must be sorted in ascending order. The array must be the same size as the number of nodes in the BST. You may not assume anything about the `BST` class or the `BSTNode` class except that it functions like a Binary Search Tree. You may write any helper methods you see fit to solve the problem.

```
public class BST{
    private BSTNode root;

    public Comparable[ ] toArray(){

    }
}
```

## 9. Hash Tables[ 10 pts ]

Given the following class `HashNode`:

```
public class HashNode{
    private Object key;
    private String data;
    private HashNode next=null;
    private void setKey(Object key){ this.key = key; }
    public void setData(Object data){ this.data = data; }
    public void setNext(HashNode next){ this.next = next; }
    public String getData(){ return data; }
    public Object getKey(){ return key; }
    public HashNode getNext(){ return next; }
    public HashNode(String data){ this.data = data; }
}
```

Write a `toString` method for the `HashTable` class, which will print out the `HashTable`. There should be a newline character between indices in the `HashTable`. The `toString` should return a `HashNode` followed by `->>` followed by another `HashNode`, etc. An example of a hash table of size 5 with 3 nodes in it.

```
null
null
OneData ->> ThreeData ->> null
null
TwoData ->> null
```

```
public class HashTable{
    private HashNode[] table = new HashNode[7];
    public String toString(){
```

```
    }
}
```

10. **Binary Search Trees**[ 10 pts ]

Given the following class `BSTNode`:

```
public class BSTNode{
    private Comparable data;
    private BSTNode right=null, left=null;
    public void setData(Comparable data){ this.data = data; }
    public void setRight(BSTNode right){ this.right = right; }
    public void setleft(BSTNode left){ this.left = left; }
    public Comparable getData(){ return data; }
    public BSTNode getRight(){ return right; }
    public BSTNode getLeft(){ return left; }
    public BSTNode(Comparable data){ this.data = data; }
}
```

Write the method **public boolean isFull()**, which returns whether or not the tree is full. You may not assume anything else about the BST class or the `BSTNode` class. You may write any helper methods you see fit to solve the problem.

```
public class BST{
    private BSTNode root;

    public int height(){
```

```
    }
}
```