PROBLEM 2: (I've got a URL Jones (20 points))

Each URL on the world-wide web has a corresponding Internet Protocol (IP) address. For example, the IP address of www.duke.edu is 152.3.233.20. The table below shows some URLs and the corresponding IP address.

<table>
<thead>
<tr>
<th>URL</th>
<th>IP-address</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.parking.duke.edu">www.parking.duke.edu</a></td>
<td>152.3.9.199</td>
</tr>
<tr>
<td><a href="http://www.pastoralleadership.duke.edu">www.pastoralleadership.duke.edu</a></td>
<td>152.3.90.245</td>
</tr>
<tr>
<td>genomics.duke.edu</td>
<td>152.3.232.39</td>
</tr>
<tr>
<td><a href="http://www.events.duke.edu">www.events.duke.edu</a></td>
<td>152.3.9.120</td>
</tr>
<tr>
<td>bigbang.phy.duke.edu</td>
<td>152.3.182.5</td>
</tr>
</tbody>
</table>

In this problem URLs and corresponding IP-addresses are stored in strings separated by one space as shown below.

"www.parking.duke.edu 152.3.9.199"
"www.pastoralleadership.duke.edu 152.3.90.245"
"genomics.duke.edu 152.3.232.39"
"www.events.duke.edu 152.3.9.120"
"bigbang.phy.duke.edu 152.3.182.5"

Part A (4 points)
Write method getIPaddress which returns the IP-address portion of a String representing a URL/IP-address pair. The URL and IP-address are separated by one space. For example, getIP("www.cnn.com 64.236.16.52") returns the string "64.236.16.52".

```java
/**
 * @param s is a URL/IP-address pair, URL separated from IP-address by one space
 * @return the IP-address portion of the String (after the space)
 */
public String getIP(String s){
```
Part B (8 points)
Write method `getAllIPAddresses` which returns an array of IP-addresses extracted from each String in the array `list` which stores URL/IP-address Strings as described above. If `list` represents the five URL/IPs Strings above, the returned array should be as shown below.

```java
{"152.3.9.199", "152.3.90.245", "152.3.232.39", "152.3.9.120", "152.3.182.5"}
```

In writing `getAllIPAddresses` you may call `getIP` from Part A. Assume `getIP` works as specified.

```java
/**
 * @param list is an array of URL/IP-address pairs
 * @return an array of just the IP-addresses represented in list
 */
public String[] getAllIPaddresses(String[] list){
    // Do Not Do This Problem
}
```
Part C (8 points)

Complete the method `findIPAddress` which returns a String representing the IP address corresponding to the URL specified by parameter `url`. The array `list` stores Strings representing URL/IP-address pairs as described above. If the URL specified by parameter `url` is not found in the array return "0.0.0.0" for the IP-address. For example, if `list` represents the data above the calls and values below help explain the method you'll write.

<table>
<thead>
<tr>
<th>call</th>
<th>return value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>findIPAddress(list,&quot;www.parking.duke.edu&quot;)</code></td>
<td>&quot;152.3.9.199&quot;</td>
</tr>
<tr>
<td><code>findIPAddress(list,&quot;genomics.duke.edu&quot;)</code></td>
<td>&quot;152.3.232.39&quot;</td>
</tr>
<tr>
<td><code>findIPAddress(list,&quot;cs.duke.edu&quot;)</code></td>
<td>&quot;0.0.0.0&quot;</td>
</tr>
</tbody>
</table>

Complete the method below. You may call methods specified in Part A and Part B. Assume they work as specified.

```java
public String findIPAddress(String[] list, String url){
```
In this problem assume that a method `convertAll` is implemented. It returns an array of all the proteins in a dna-sequence (you may have written a similar method for extra credit, and we discussed a similar method in class.) The returned array may have zero proteins or 100 proteins. In the version of `convertAll` below, an ORF is specified.

```java
/**
   * Returns all proteins found in dna for specified orf
   * @param dna is the sequence from which proteins are obtained
   * @param orf is the open reading frame used in finding proteins
   */
   public String[] convertAll(String dna, int orf){
       // implementation not shown, but convertAll can be called
   }
```

**Part A (6 points)**

One part of determining the best ORF for a protein is the number of proteins coded using the ORF. Write a method `bestORF` which returns which of the ORFs 1, 2, or 3 is the best for a particular DNA sequence. The best ORF is the one for which the most proteins are found. You may call `convertAll` shown above, assume it works as specified.

```java
/**
   * Returns the best ORF for dna, where best decided by which ORF codes the most proteins
   * (considering only ORFs 1,2,3)
   * @param dna for which the best ORF is found
   * @return the best ORF
   */
   public int bestORF(String dna){
   }
```
Part B (4 points)

An alternative definition of best ORF is based on which ORF produces a group of proteins with the maximal median length. So, an ORF which codes proteins whose lengths are 20, 20, 50, 60, 70 (median length is 50) is better than an ORF which codes proteins whose lengths are 10, 10, 15, 15, 20, 20, 30, 30, 50, 50, 70 (median length is 20). The median is the middle value of a sorted list.

Don’t write code, but describe the code/method you’d write to determine the best ORF using these criteria. Be precise, but don’t write code.
PROBLEM 4: (Complementary Proteins 18 points)

In this problem you’ll need to use the class ProteinTool shown below by calling the static methods appropriately in solving problems related to protein hydrophilicity – the extent to which proteins have an affinity for water.

public class ProteinTool
{
    /**
     * @param p is the coding for a protein, e.g., "LWAGGYK"
     * @return the hydrophilicity index of the protein coded by p
     */
    public static double hydrophilicity(String p){
        // implementation not shown
    }

    /**
     * Return Hopp Woods Hydrophilicity index for amino-acid p
     * @param p represents an amino acid, e.g., "W", or "K" or ...
     * @return the appropriate hydrophilicity value
     */
    public static double hoppWoodValue(String p){
        // implementation not shown
    }
}

no code to write on this page
Part A (6 points)
Write the method `hydroCount` which calculates the number of proteins whose hydrophilicity is above a specified threshold and returns this number. The array parameter `plist` is an array of proteins, parameter `thresh` is the minimal value for which a protein should be considered hydrophilic. For example, if `plist` stores 10 proteins whose hydrophilicity indexes (as obtained from `ProteinTool.hydrophilicity`) are as follows:

```
2.0 1.5 2.2 1.7 2.3 -1.5 -3.0 -2.0 1.5 1.8
```
then the call `hydroCount(plist,-2.5)` should return 9, the call `hydroCount(plist,2.1)` should return 2 and the call `hydroCount(plist,2.8)` should return 0. In writing `hydroCount` you’ll need to call `ProteinTool.hydrophilicity` to determine a protein’s hydrophilicity.

```java
/**
 * Return number of proteins in plist with hydrophilicity above thresh
 * @param plist is an array of proteins
 * @param thresh is the hydrophilicity threshold
 */
public int hydroCount(String[] plist, double thresh){
```

```java
}
```