Problem 1: [16 points]

Below is some mysterious code involving sets. Describe, in English, what the method does. What would be a more descriptive name for the method?

```
public static Set mystery(Set s1, Set s2) {
    HashSet results = new HashSet();
    for(Object item : s1) {
        if (!s2.contains(item)) {
            results.add(item);
        }
    }
    for(Object item : s2) {
        if (!s1.contains(item)) {
            results.add(item);
        }
    }
    return results;
}
```
Problem 2: [20 points]

You’ve been asked to help implement a thesaurus application that uses a HashMap to map from words to corresponding lists of synonyms. Part of the code is shown below. It’s your job to finish the definition of the addWord method, which takes a word and a synonym and adds the synonym to the map. For example, if passed “fast” and “swift”, the method would add “swift” to the list of synonyms for “fast”. Use the back of this page for your solution if you’d prefer.

```java
public class Thesaurus {
    private HashMap<String, ArrayList<String>> words =
        new HashMap<String, ArrayList<String>>();

    public void addWord(String word, String similar) {
```

```java
```
**Problem 3: [24 points]**

**Part a)** Below, draw the Binary Search Tree (BST) that would result from inserting the following sequence of values into an initially-empty BST: 50, 30, 80, 40, 45, 35, 90, 60, 85. (That is, first insert a 50 into an empty tree, then insert the 30 into the tree, etc.)

**Part b)** Draw the BST that would result from removing the 30 and 45 from the tree above.

**Part c)** Is it possible to create a *perfect* BST that contains the values from Part b? If so, draw the tree. If not, explain why not.
Problem 4: [40 points]

You’re doing a whiteboard interview with a recruiter from a famous tech company, and she asks you to analyze four different algorithms for solving a matching problem: Given an array of integers as input, determine which numbers have a corresponding “mate” in the array with which it would sum to 10. More specifically, you should print out “yes” or “no” for each item in the array — “yes” if that item has at least one mate in the list that could be added to it to produce 10, and “no” if no such mate exists. For example, if the input was [2,7,2,8,5], the output should be “yes”, “no”, “yes”, “yes”, “yes”. We print “yes” for the 2 because the 8 in the array could add with it to produce 10. There’s no such mate for 7 so we print no. That last “yes” is because we’re allowed to add an item to itself when trying to reach 10.

a) The first proposed algorithm would loop through and consider each item in the array in turn. For each item, another pass would be made to look for its mate. For example, when considering the 2 at the start of the array, we’d make a pass starting at the beginning and consider the 2, the 7, the 2, and then the 8 as mates. When we got to the 8 we’d stop the search and print “yes”. If we get all the way through the array without finding a mate we print “no” and then move on to consider the next value from the array. For an array of size n, what’s the Big-O complexity of this algorithm? (You don’t need to show a detailed T(n) function as part of your work, but for full credit you should justify your Big-O answer briefly.)

b) The second algorithm starts by making a pass over the array, copying each of the items into a HashSet of Integers. It then makes a second pass to consider each item. During the second pass, we calculate the number’s mate and check to see if it’s in the set. For example, when we consider the 2, we calculate that its mate must be 8 (that’s 10-2), and we’d print “yes” since the set’s contains method returns true for an 8. For the 7 we’d calculate that its mate would be a 3, then print “no” when we discover that there’s no 3 in the set, and so on. For an array of size n, what’s the Big-O complexity of this algorithm?
c) The third algorithm first sorts the array using Merge Sort, then makes a pass over the sorted array to consider each value and print the corresponding “yes” or “no”s. When considering each number we calculate the value of its mate as we did in the previous algorithm, but now we use binary search on the sorted array to see if the mate is in the array. For an array of size n, what’s the Big-O complexity of this algorithm?

d) The final algorithm makes an initial pass over the array and stores copies of the values in a heap. We then iterate over the values from the heap: While the heap isn’t empty, take the value off the top of the heap and go look for its mate in the original array (starting at the front and working through as in part a). Print “yes” if we find it, “no” otherwise, then repeat until the heap is empty. For an array of size n, what’s the Big-O complexity of this algorithm?