1. Given a binary search tree class (BST) with the following definition:

```
public class BST<T extends Comparable> {

    Node<T> root;
    int count;

    private class Node<T extends Comparable>  {
        T value;
        Node left;
        Node right;

        public Node(T value) {
            this.value = value;
            left = right = null;
        }
    }
}
```

Assume that there are standard implementations of:

```
public BST();                     // Default constructor creates an empty BST
public boolean insert(T data);    // Inserts a node containing 'data' into BST,
                                  //   returns TRUE on success
public T delete(T data);          // Removes a node containing 'data' from BST,
                                  //   returns 'data'
public boolean contains(T data); // Returns TRUE if 'data' is in the BST
```

Implement the following methods:

(a) [8 points] `public int countLeaves();` - returns the number of leaves in the BST

(b) [8 points] `void printSingleBranches();` - outputs all nodes with exactly one (1) child, in ascending order

(c) [8 points] `void reverseOrder();` - outputs all the nodes in the tree in descending order

2. Given the following hashing class that implements linear probing, with *dummy values* inserted for removed entries:

```
public class LinearHash {

    String []table;  // Hash Table Entries
    int used;        // Count of used entries
    int probeDist;   // Probing distance (default: 1)
```

Assume that there are standard implementations of:

```
public HashTable(int size, int probDist);  // Constructor for HashTable with 'size
' entries
                                           //   and probing distance 'probDist'
public HashTable(int size);          // Constructor for HashTable with 'size'
entries
                                           //   and probing distance: 1
```

Implement the following methods:

(a) [7 points] `public double load();` - returns the current *load factor* for the table.

(b) [7 points] `public int insert(String value);` - adds `value` to the table, returns the index

(c) [7 points] `public int find (String value);` - returns the index of `value` in the table

(d) [7 points] `public String delete(String value);` - removes value from the table, and returns it.

Note: `delete()` should place a *dummy value* of `""` into the table to represent a removed entry. `insert()` and `find()` should also do the "right thing" with the dummy values.

3. [9 points] Implement the following static method that utilizes the Set Interface:

```
public static Set intersection(Set a, Set b);
```

This method should return a `Set` that contains only those `Objects` that appear in both `Set a` and `Set b`. The returned `Set` must only meet the `Set` Interface. Your implementation is free to use classes included in the base Java SDK, but not from other packages and libraries.

4. You have been asked to design and implement a (mini) "spell checker" application. Your application will need to process a file of correctly spelled words, and another file which is the document (text file for our case) that you want to perform spelling check.

For this problem only - you may utilize any data structures available from the Java Platform API (such as list, map, etc.)

(a) [8 points] In a paragraph or two, discuss how you would solve this program. In particular, explain the data structure(s) that you are planning to use, and why it's your best choice(s).

(b) [18 points] Provide a Java implementation that will:

1. Read a file, called "wordlist.txt" of correctly spelled words; one word per line, all words will be lowercase.
2. Process the "input.txt" file; this is the file that you want to spell check. Assume the entire file will be lowercase, without any punctuation, but multiple words may appear on each line.
3. Identify and display all of the misspell words in "input" file; A word is misspelled if it does not exist in the "wordlist.txt" file.
4. In the event a file cannot be found (*ie.* `FileNotFoundException`), your code should print a meaningful error along with the name of the offending file. Your program should then exit cleanly (*eg.* no stack trace printed)

5. Given the following sequence of integers: 67, 89, 100, 83, 50, 55, 42, 95, 22, 66

(a) [5 points] Show the binary search tree built from this sequence of integers.

(b) [3 points] Show the pre-order traversal

(c) [3 points] Show the in-order traversal

(d) [3 points] Show the post-order traversal

(e) [6 points] Represent (show) the tree if stored as an array representation

6. Determine the algorithmic complexity of the following methods:

(a) [3 points]
```
public static void recurse1(long n) {
  if (n > 0)
    recurse1(n-1);
}
```

_____/86 points

(b) [3 points]
```
public static void recurse2(long n) {
    for (long i = 0; i < n; i++)
    {
        recurse2(n-1);
    }
}
```

(c) [3 points]
```
public static void recurse3(long n) {
  if (n > 0) {
    recurse3(n-1);
    recurse3(n-1);
  }
}
```

7. Given the following *min-heap* in array form:

2	19	25	38	55	60	40	53	80	87	85

   (a) [6 points] Add the value 23 to the heap, and then show the new min-heap in array representation.

   (b) [6 points] Remove the value 19 from the heap you created in question 7a, , and then show the new min-heap in array representation.

8. [12 points] Assume the BST class from question 1 contains a `boolean equals(BST b)` method which will return `true` if *in-order* traversals of the trees are the same. Create a `int hashcode()` method that generates hashcodes that conform to the Java hashcode contract. Feel free to create any support methods you need to complete the problem.