**Programming Assignment#2 – COP3503H Fall 2011**

**Due: Wednesday, September 21 at 11:59PM**

Write a Java program to implement the ADT (interface) **Partition**:

```
public interface Partition {

/*
 * Unions the partition containing element1 with that containing element2.
 * Reduces the number of partitions by one, provided the elements are not
 * already in the same partition.
 */
    void union(int element1, int element2);

// Returns the "representative element" in the partition containing element.
    int find(int element);

}
```

You will the need to implement this in two different classes, each of which inherits from a class Partition that I wrote. Partition implements AbstractPartition. It implements

**public Partition(int n)** – Constructs the Partition having **n** partitions, designated by the integers **0…n-1**.

**public String toString()** – Provides a printable report depicting current partitions.

**public String statString()** – Provides a printable report depicting total number of calls to find, and total and average number of elements inspected to perform the find service.

**public void resetStats() –** Resets number of finds and number elements inspected during finds.

Of course, it also implements union and find, as required by the AbstractPartition contract.

Test your class by using variants of the following Java main program.

```
package COP3503H;

import java.io.*;
import java.util.*;

class PartitionTest {

  public static void main(String[] args) throws IOException {
    for (int SZ=16; SZ<=1024; SZ <<= 2) {
       Random r =  new Random(168886542);
       Partition p = new Partition(SZ);
       System.out.println(p.toString());
       for (int i = 0; i < SZ-1; i++) {
          int rep1, rep2;
          do {
             int el1 = Math.abs(r.nextInt())%SZ; int el2 = Math.abs(r.nextInt())%SZ;
             rep1 = p.find(el1); rep2 = p.find(el2);
          } while (rep1 == rep2);
          p.union(rep1,rep2);
       }
       System.out.println(p.toString());
       p.resetStats();
       for (int i = 0; i < 10000; i++) p.find(Math.abs(r.nextInt())%SZ);
       System.out.println(p.statString()+"\n");
    }
    System.out.println("**** Press Enter ****"); System.in.read();
  }

}
```

We really want to run this three times: once with Partition, once with SmartPartition and once with CompressedPartition. The first, which I already implemented, is the totally naive union algorithm that ignores depths; the second minimizes the depth of the unioned tree; and the third uses path compression during find operations. Your SmartPartition must extend my partition, and your CompressedPartition must extend your SmartPartition.

Now, run all three implementations on partitions of size (SZ) 16, 64, 256, 1024. Report the average you got from the 10000 random finds exercise of the structure. Analyze these results, comparing them to the theoretical expectations. (A nicely formatted table is required here.)

Send me a zipped copy of the report and all source files necessary to test your program.