```java
/* Problem 5--Slow Sorting
   This is a classic algorithm.  While you sort the array with Merge
   Sort, you count the inversions at each step.  Since the data cases
   were very large (several were a million in length) if you counted
   the inversions one at a time, you'd run out of time.  Also, you
   need to store the answer in a long. */

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

public class prob5 {

 private static Scanner in;
 private static PrintWriter out;
 private static int cs;
 private static String[] Words;

 public static void main (String[] args) throws Exception {

  in = new Scanner (new File ("prob5.in"));
  out = new PrintWriter ("prob5.out");
  cs = 1;
  while (true) {
   int len = in.nextInt();
   if (len==0) break;
   in.nextLine();
   Words = new String[len];
   for (int i=0; i < len; i++) Words[i] = in.next();
   out.printf ("Case %d: You would have to exchange at least %d "+
               "adjacent pairs.\r\n\r\n",cs++,invcount(Words));
  }
  in.close();
  out.close ();
 }

 public static long invcount (String[] A) throws Exception {

  if (A.length <= 1) return 0;
  String[] B = new String[(A.length+1)/2];
  String[] C = new String[A.length/2]; /* Split list in half */
  for (int i=0; i < B.length; i++) B[i] = A[i];
  for (int i=0; i < C.length; i++) C[i] = A[B.length+i];
  long ic = invcount (B)+invcount(C); /* Recursively sort and count */
  int j=0, k=0; /* Start the merge of the sorted lists */
  for (int i=0; i < A.length; i++)
   if (k==C.length || j < B.length && B[j].compareTo(C[k]) <= 0)
    A[i]=B[j++]; /* If next element is from the first list, good */
   else { /* If next element is from the second list */
    A[i]=C[k++]; /* add to the count all the elements that element */
    ic += B.length-j; /* would skip over */
   }
  return ic;
 }
}

```