3.

CODE:

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

public class Solution{
private BufferedReader in;
private StringTokenizer line;
private PrintWriter out;

private static final int mm = 1000000007;
public void solve() throws IOException {
int[] a = nextIntArray(3);
int[] b = nextIntArray(3);
int aa = 0;
int bb = 0;
for(int I = 0; I < 3;I++){
  if(a[i])>b[i]) aa++;
  else if (a[i] < b[i]) bb++;
}
  System.out.println(aa+””+bb);
}
public static void main(String[] args) throws IOException{
   new Solution().run(args);
}
public void run(String[] args) throws IOException{
   if(args.length > 0 && “DEBUG_MODE”.equals[0])){
     in = new BufferedReader(new InputStreamReader(new FileInputStream("input.txt"))); }
   else { in = new BufferedReader(new InputStreamReader(System.in));
```

```
} out = new PrintWriter(System.out);
int t = 1;
  for (int i = 0; i < t; i++)
 { // out.print("Case #" + (i + 1) + ": ");
  solve(); }
 in.close();
out.flush();
 out.close(); }
private int[] nextIntArray(int n) throws IOException {
 int[] res = new int[n];
  for (int i = 0; i < n; i++) {
   res[i] = nextInt(); } return res; }
private long[] nextLongArray(int n) throws IOException {
   long[] res = new long[n];
 for (int i = 0; i < n; i++) {
  res[i] = nextInt(); }
  return res; }
private int nextInt() throws IOException {
   return Integer.parseInt(nextToken()); }
private long nextLong() throws IOException {
  return Long.parseLong(nextToken()); }
private double nextDouble() throws IOException {
  return Double.parseDouble(nextToken()); }
private String nextToken() throws IOException {
 while (line == null || !line.hasMoreTokens()) {
line = new StringTokenizer(in.readLine()); }
  return line.nextToken(); }
private static class Pii {
private int key;
private int value;
```

```
public Pii(int key, int value) {
this.key = key;
this.value = value; }
@Override public boolean equals(Object o) {
  if (this == o) return true;
    if (o == null || getClass() != o.getClass())
      return false;
Pii pii = (Pii) o;
  if (key != pii.key)
     return false;
    return value == pii.value; }
@Override public int hashCode() {
int result = key;
result = 31 * result + value;
   return result; }
@Override public String toString() { return "Pii{" + "key=" + key + ", value=" + value + '}'; } }
private static class Pair<K, V> {
private K key;
private V value;
public Pair(K key, V value) {
   this.key = key; this.value = value; }
public K getKey() {
   return key; }
public V getValue() {
  return value; }
@Override public boolean equals(Object o) {
if (this == o) return true;
if (o == null || getClass() != o.getClass())
   return false;
Pair<?, ?> pair = (Pair<?, ?>) o;
```

```
if (key != null ? !key.equals(pair.key) : pair.key != null)
  return false; return
!(value != null ? !value.equals(pair.value) : pair.value != null); }
@Override public int hashCode() {
int result = key != null ? key.hashCode() : 0;
result = 31 * result + (value != null ?
value.hashCode() : 0);
  return result; }
 }
 }
```