Solution notes

Data Structures and Algorithms 2005 (MR)
Paper 6 Question 1, Paper 13 Question 1

This question draws on basic knowledge of many algorithms covered in the course, together with an ability to reason about algorithm costs.

(a) Vector count[1] ... count[N] holding the counts. A vector index[1] ... index[N] holding counter numbers in increasing order of value. So count[index[1]] in the smallest counter value (=mincount) and count[index[N]] is the largest counter value (=maxcount). A vector inverse which is the inverse of index, ie index[inverse[i]]=i so that when increment(i) is called it can find which element of index is affected. When increment(i) is called, count[i] is incremented and then the elements of index and inverse possibly changed to maintain the ordering. Some adjacent elements of index may have to be swapped, but not many if N is known to be about 10. Alternatively, do not keep the counters in sorted order but look through all N counters when one with the minimal value is incremented to see if mincount has to change. If a counter with value maxcount is incremented then so is maxcount.

(b) If N is about 10**6 the pairwise exchanges may become too expensive. For instance incrementing counter 1 when all counter are zero would cause 10**6-1 exchanges. This could be much improved by hold the counts in a priority queue based on a binary heap as in heapsort. count[index[1]] would still be mincount, but otherwise the counters would not be fully sorted. The number of swaps needed when a counter is incremented would now be no more than log2(N) (~=20) but typically much less. If the incremented counter is larger that maxcount, maxcount would be updated. If count[index[1]] changes, mincount would be updated. Alternatively, if maxcount-mincount is known not to be too large, the following scheme could be used. Allocate a vector v of integers somewhat larger than the expected maximum difference of maxcount and mincount. This vector will be used as a circular buffer containing the count of how many counters have each possible value. v[mincount mod size] will be the number of counters with value mincount, v[maxcount mod size] will be the number of counters holding value maxcount, etc. In general, v[count[i] mod size] will hold the number of counters with value count[i]. When counter i is incremented v[count[i] mod size] is decremented and v[(count[i]+1) mod size] is incremented. With this scheme it is easy to determine when to increment mincount. Dealing with maxcount is already easy.

(c) At the end of the run the average value of each counter will be about 1000. So halfway through the run they will have values aroud 500. In a sorted list of these count values we can expect long sequences of equal values, probably in the region of 2000 to 10000 in length. We can thus expect the number of swaps per counter increment when using algorithm (a) to between 1000 and 5000, compared with probable 2 or 3 swaps needed for algorithm (b) The improvement therefore somewhere around 1000/3 to 5000/2, or 300 to 2500. Certainly significant. Better students may even produce more accurate estimates, but probably not many.