Problem 1

a) The baskets that contain {3, 4} are 12, 24, 36, 48, 60, 72, 84, 96. Hence, the support for {3, 4} is 8.

b) All the frequent items are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

c) {1, 2, 4} has the highest support, which is 25.

d) Yes. The general format of an association rule that has 100% confidence is $\{i_1, i_2, \ldots, i_k\} \rightarrow y$, where y is a factor of the least common multiple (LCM) of $\{i_1, i_2, \ldots, i_k\}$. The following are some examples:
   1. $\{4\} \rightarrow 2$, where 2 is a factor of 4.
   2. $\{2, 3\} \rightarrow 6$, where 6 is not a factor of either 2 or 3, but a factor of the LCM of {2, 3}, i.e., 6.

Common Mistakes:

a) 1A [-2] Give something relevant, but not the support, which is the number of baskets containing {3, 4}.
   1B [-5] Give an incorrect support without any explanation.

b) 1C [-2] Give the frequent item sets, instead of the frequent items.

c) No common mistake.

d) 1D [-2] Give an example with a partially-incorrect explanation.
   1E [-3] Give an example without any explanation.
   1F [-5] Answer no.

Problem 2

The following is the layout of counters of pairs in an array (for n = 6):

i / j	1	2	3	4	5
0	0	1	2	3	5
1		5	6	7	8
2			9	10	11
3				12	13
4					14

Notice the following points about the above layout:

1. The number of entries in the array is ${}_nC_2$, i.e., n(n – 1) / 2.
2. The counter for a pair {i, j} where i = j is not included in the array since only the pairs of different items are relevant in the a-priori algorithm.

Given a pair {i, j} where i ? j, the array index of its counter can be calculated as follows:

1. Let x = min(i, j) and y = max(i, j).
2. index = (x * n) + ((x + 1) * x / 2) + (y – x – 1). (x * n) can be interpreted as the starting index of a row containing x. ((x + 1) * x / 2) is to account for the fact that no counter is allocated to a pair {x, y} where x ? y. (y – x – 1) is the offset within the row containing x.

Common Mistakes:

2A [-3] Not handle the requirement that a pair must be associated with the same array entry regardless of the order of the items.

2B [-3] Include {i, j} where i = j in the array. It should be not included since it is irrelevant in the a-priori algorithm.

2C [-1] Give a method for calculating the array index with an off-by-one error.

2D [-3] Give a partially incorrect method for calculating the array index.

2E [-6] Not give a method for calculating the array index.

2F [-10] Propose another less efficient data structure.

2G [-20] Not give the details about the method for assignment array entries to pairs.

Problem 3

In the first pass, a counter is kept for each item and only an array of 1,000,000 integers is needed. Therefore, the memory required for the first pass is 4 bytes * 1,000,000 = 4,000,000 bytes.

The results of the first pass are summarized as an array of the labels of frequent items in the second pass. The memory required is 4 bytes * 25,000 = 100,000 bytes.

Two ways of organizing the counters of potentially frequent pairs in the second pass are as follows:

1. The first is to store the counters of all the potentially frequent pairs and an array of ${}_{25000}C_2$ integers is needed. Notice that their labels should not be stored as a part of the array. Therefore, the memory required for the second pass is 100,000 bytes + 4 bytes * (25000 * 24999 / 2) = 1,250,050,000 bytes.
2. The second is to store the counters of the potentially frequent pairs that occur at least once and an array of (1,000,000 + 100,000,000 / 2) entries, each of which consists of a label (8 bytes) and a counter (4 bytes), is needed. Therefore, the memory required for the second pass is 100,000 bytes + (8 + 4) bytes * (1,000,000 + 100,000,000 / 2) = 612,100,000 bytes.

The second way of the second pass requires less memory. Hence, the total memory required is max(100,000, 612,100,000) = 612,100,000 bytes = 583.74 MB.

Common Mistakes:

3A [-4] Ignore the first pass to calculate the total memory required. There is no guarantee that the second pass requires more memory than the first pass. Hence, both passes should be taken into account.

3B [-2] Store the counters of all items, instead of just the labels of frequent items in the second pass.

3C [-2] Store both of the new and old labels, instead of just the old labels, of frequent items in the second pass.

3D [-2] Store both of the labels and counters, instead of just the counters, of pairs of frequent items in the first method of the second pass.

3E [-2] Store just the counters, instead of both of the labels and counters, of pairs of frequent items in the second method of the second pass.

3F [-2] Use an incorrect data structure.

3G [-2] Use an incorrect figure, e.g., the number of entries in the second pass.

3H [-6] Give an inaccurate memory requirement for the second pass.

3I [-2] Take the sum, instead of the maximum, of the memory required in the first pass and that in the second pass.

3J [-2] Not give the total memory required.

Problem 4

Let b = number of buckets of the hashtable in the first pass.

In the first pass, a counter is kept for each item and for each bucket in the hashtable. Therefore, the memory required for the first pass is 4 bytes * 1,000,000 + 4 bytes * b = (4 * 10^6 + 4 * b) bytes.

The results of the first pass are summarized as an array of the labels of frequent items and a bitmap of the hashtable in the second pass. The memory required is 4 bytes * 25,000 + b / 8 bytes = (10^5 + b / 8) bytes.

The number of candidate pairs is 1,000,000 + 50,000,000 * 1,000,000 / b. An entry, which consists of a label (8 bytes) and a counter (4 bytes), is kept for each candidate pair. Therefore, the memory required for the second pass is (100,000 + b / 32) bytes + 12 bytes * (1,000,000 + 50,000,000 * 1,000,000 / b) = (1.21 * 10^7 + b / 32 + 6 * 10^{14} / b) bytes.

To minimize the total memory required, we have to solve the following equation:

$$4 * 10^6 + 4 * b = 1.21 * 10^7 + b / 8 + 6 * 10^{14} / b$$

b = 13,532,397 and the total memory required is (4 * 10^6 + 4 * 13,532,397) bytes = 58,129,588 bytes = 55.44 MB.

Common Mistakes:

4A [-2] Use an incorrect data structure to count items in the first pass.
4B [-3] Not count items in the first pass.
4C [-5] Choose the number of buckets in the hashtable randomly.
4D [-2] Use an incorrect data structure to store the results of the frequent items in the second pass.
4E [-3] Not store the results of the frequent items in the second pass.
4F [-2] Use an incorrect data structure to store the results of the hashtable in the second pass.
4G [-3] Not store the results of the hashtable in the second pass.
4H [-5] Calculate the number of candidate pairs in the second pass incorrectly.
4I [-2] Use an incorrect data structure to store candidate pairs in the second pass.
4J [-2] Use an incorrect data structure for the hashtable in the first pass.
4K [-20] Misunderstand the PCY algorithm.